MSX FAN SERIES 2 マシン語入門（応用編）

MIA

MSX FAN SERIES

2

マシン語入門 応用編

MSX ハード活用術

マシン語ゲームをつくる



まえがき

ホーム・パソコンの標準規格として，ＭＳＸが発表された当初は，その性能等についていくつか評価がわかれていましたが，その後ゲームを中心としたソフトウェアが続々と登場し，すっかりアミューズメント志向の低価格マシンという形で定着してきたようです。

ＭＳＸに代表されるパーソナル・コンピュータの利用法には，ゲーム・ソフトを購入して楽しむ，BASICを勉強するなど，いろいろと考えられます。ところで，もしあなたがいつも一歩先を目指す人ならば，ここで思い切って自分のオリジナル・ゲームをつくってしまうというのはどうでしょう。

音楽マニアが高じてミュージシャンになってしまったという例もありますが，“創る”ことは受け身として単に遊ぶのと比べ，はるかに楽しく，すばらしい何物かを得られます。

こういったクリエイティブなパソコン・ライフを志向する人びとのために，本書はソフトウェア制作等の手助けになりうると確信しています。

本書では『ＭＳＸでマシン語ゲームをつくる』ということを目的に，ＭＳＸのハードウェアの具体的な活用法，およびゲーム制作のポイント等について解説していきます。したがって，マシン語によるプログラム例も多数掲載されていますが，とくにマシン語の学習書というわけではありません。Z80のマシン語については，前書にあたる『ＭＳＸマシン語入門(基礎編)』程度の知識を備えてあれば問題ありません。

また，本文中のソース・リストはＢＡＳＩＣのエディタで作成したもので，これは基礎編に掲載されているアセンブラでアセンブルできます。もっともこのアセンブラは標準的なものなので他のアセンブラでも代用できるでしょう。

今のところゲームをつくるプログラマーは，マーケットの大きさに較べて非常に少なく，なかなか貴重な存在だと言えます。本書がきっかけとなって，新たなプログラマーが一人でも発掘されることになれば幸いです。

1984年　9月　白井康之

第1章 マシン語によるゲーム作り
ゲーム制作に挑戦しようというあなたへ

第2章 MSXのグラフィック機能を使いこなす
VDP(TMS9918A)の使いかた

1

1章 マシン語による ゲーム作り

ゲーム制作に挑戦しようというあなたへ

本章では，パソコン・ゲームの分類を通じて，優れたゲームを制作するためには何が必要か解説していきます。

マシン語によるゲーム作り

1 スタープログラマーをめざせ

中学生に『今，一番欲しいものは何か。』という質問をすると，『パソコンが欲しい。』という答が返ってくるそうです。数年前までは，ごく一部のマニアの趣味に過ぎなかったパソコンが，こうも早く市民権を得てしまうとは，時代の変化の速さに改めて驚かされてしまいます。

ところで急速に普及したパソコンですが，いったいどんな使われ方をしているのかというと，日本では主にホビーユースでゲームをして遊ぶ，という人が大部分のようです。一方，日本よりも２年進んでいるといわれるアメリカでは，若干様子が異なり，ゲームの伸びはむしろ下火で，ビジネス・ソフトなどの伸びが目立ちます。

こうしてみると，日本のゲーム・ソフトの将来は暗い，というように思われがちですが，こう決めてしまうのはまだ早計です。アメリカの場合でもゲーム・ソフトの市場はむしろ拡大しており，実際には成熟期を迎えているといった表現の方が適切だと言えるでしょう。ただし，優れたゲーム以外は次第に淘汰されているようです。

正直なところ，日本のゲーム・ソフトはまだ玉石混合の状態ですが，将来アメリカのように市場が成熟してくると，こうした“石”のソフトは，どんどん消えていくことになるでしょう。結局優れたモノだけが残るというわけです。

ところで話は飛ぶのですが，最近『ゲーム作家』なるものが巷で注目を集めているようです。10代の後半から20代の前半くらいで何千万円も稼いだ，などという話を聞くと『ボクのところにも少しぐらいはおこぼれがあってもいいはずなんだけどなぁ。』と思わず自分の限りなく軽い財布を呪ってしまいます。もっとも，実際に何千万円も稼いでいるという人はほんの僅かで，独立して商売としてやっていくのはなかなか大変なようです。

しかし，パソコンとより進んだ形でつき合おうとしているなら，一度くらいスタープログラマーを目指してみたいものです。スタープログラマーは今やカッコイイ人種の代名詞，あなたが男性であれば，どこへ行っても女の子にモテモテのはずです。たとえば，

あなた『キミ，ボクのＢＭＷでドライブしないか。』

彼女　『ええ，喜んで。でもあなたってお金持ちネ，この車どうして

手に入れたの？』
あなた『ふっふっふ，いやちょっとしたゲームをつくってねェ，その
印税を頭金にして，ローンで買ったのさ』
彼女　『ダサイわ』
あなた『……』
ということになるかどうかはわかりませんが，とにかくあなたもゲームづくりに挑戦してみてはどうでしょうか。つくったゲームが優れたものであれば，確実に実入りはあるはずです。

マシン語によるゲーム作り

2 パソコン・ゲームの分類

パソコン・ゲームとひと口に言っても，その内容はさまざまで，反射神経がものをいうもの，精神集中を必要とするもの，果てはゲームが終了するまで半年以上かかるなんてものまであります。そこで現在売られているパソコン・ゲームを，それぞれのタイプに分類してみることにしましょう。

1. リアルタイム

パソコン・ゲームの中で最も普及しているのが，このタイプのゲームです。インベーダー，ギャラクシアン，パックマンにゼビウスといったような，ゲーム・センターに置かれているもの（アメリカではこれらをまとめてアーケード・ゲームという）は，ほとんどこのジャンルに属しています。

こうしたゲームでは，瞬間の判断力とキーボードを叩く速さが，高得点を生むポイントになります。つまり，リアルタイム・ゲームの命は，そのスピードにあるといえるでしょう。

ところでこのリアルタイム・ゲームですが，何百種類ものゲームが出回っているため，最近ではマンネリ化の傾向にあり，なかなか『これは！』という作品にお目にかかりません。一方ではやりすたりが激しく，特にストーリーに発展性のないものは，すぐに飽きられてしまいます。そういった中で，ヒット作を作り出すためには，ストーリー，デザイン，サウンド，技術力のすべてにわたって，最高水準の出来栄えが要求されてくるでしょう。

2. アドベンチャー

アドベンチャーゲームでは，ゲーム自体が冒険小説のようなストーリーを持ち，プレイヤーがその主人公になります。ゲームを始めると，まず自分の回りの状況が絵や文字などで示され，その様子に応じて何をするかを入力しながらゲームを進めていきます。

リアルタイム・ゲームと違って，ゆっくり腰を据えて取り組まないといけないので，気の短い人やいつも忙しそうにしている人などにはウケが悪いというのが難点です。また，小説と同じように一度最後までいってしまうと，もうそれでオシマイというのも他のゲームと違った特徴だといえるでしょう。

ところで，今後こうしたアドベンチャーゲームを作ろうとするのであれば，外部ファイルとしてディスク・ドライブの使用を前提とすべきだと思います。アドベンチャーやロールプレイングといったタイプのゲームでは，画面情報や各種パラメータなど大量のデータを必要とするため，大容量で高速のディスクが不可欠になります。今のところ，まだディスク・ドライブの普及率が低いので，カセット版のアドベンチャーがかなり出回っているようですが，カセット版のものは操作性や速度の点で今ひとつで，またディスクのようにランダム・アクセスができないので，ストーリーの展開も制約されます。アドベンチャーゲームが広まるためにも，ディスクの普及を期待したいところです。

3. ロールプレイング

ゲームを構成するキャラクタの役割にそれぞれ取り決めがあって，その特性をうまく利用してゲームを進めるのがロールプレイング・ゲームです。実際のゲームではアドベンチャーの要素もかなり含まれますが，冒険の目的を達成しても必ずしも終わりにならないこと，冒険よりもキャラクタに主眼が置かれているところが違います。

4. シミュレーション

ウォーゲームやマネージメント・ゲームなどに代表されるように，設定にしたがったモデルをつくって，それを操作しながら進めるのがこのシミュレーション・ゲームです。乱数や計算が多用されるので，コンピュータに適したゲームとして，わりと古くから存在しました。かってのマイコン・マニアのあこがれの的だった「スタートレック」や，現実にパイロットの訓練用に使われているフライト・シミュレータなども，このシミュレーション・ゲームの仲間です。

シミュレーション・ゲームでは，プレイヤーがあたかも大企業の社長や，飛行機のパイロットになったかのようにふるまえるというとこ

ろに特徴があります。内容もより高度で専門的になるほどおもしろくなりますが，逆にあまり高度すぎると一般的でなくなってしまいます。一般のゲームとして作る場合には，この辺のバランスをうまく取ることがカギになってくるでしょう。

5. パズル

ジグソーパズルのような絵合わせからルービック・キューブのようなものまで，世の中には数多くのパズルが存在します。そうした中には，数学者が取り組んでもなかなか解けないようなものもあって，知的興味を満足させる格好の素材となるのがこのパズルです。

こうした既存のパズルをコンピュータ・ゲームに応用する場合には，特許などに触れていないかを事前によく確かめておく必要があります。

6. 麻雀etc.

トランプや花札それに麻雀などをコンピュータ・ゲームにアレンジしたものは，オリジナリティーこそ欠けるものの，根強い人気があります。とくに麻雀は，毎月，相当数のゲームが発表される中で，常に売り上げの上位に食い込んでいて，日本人の麻雀好きがここでも立証されているようです。

こうしたゲームのうち，コンピュータが人間の相手をしてくれるものは，コンピュータ側の手を考える『思考ルーチン』が重要になります。この思考ルーチンを一朝一夕で作ることはとても無理で，長い間のノウハウの蓄積が必要です。

こうして見ると，ゲームにはたくさんの種類があることがわかります。もちろんアイデア次第では，もっと別なタイプのゲームを作り出すこともできるでしょう。いずれにせよ，どれを選択するかはあなたの自由です。

マシン語によるゲーム作り

3 優れたゲームをつくるためには

かってマイクロプロセッサが世に登場したとき『10数年前，ひと部屋を占領していたコンピュータが数ミリ四方に収まった。』という表現をされました。たしかに10年前の目から今日のパソコンを見れば，その進歩には驚くべきものがあります。しかし，驚異的な進歩をとげたパソコンといえども，その心臓部はたかだか８ビットのＣＰＵであり，能力には限界があります。

したがってソフトウェアがいかに頑張ろうとも，ハードウェアによって定められたこの限界を超えることはできません。ただし，別な見方をすれば，こうしたハードの限界にどれだけ近づいているかがソフトの評価の目安になる，ということでもあるわけです。

コンピュータにＡＰＰＬＥというマシンがあります。このAPPLEはパーソナル・コンピュータという言葉が生まれた当初からあり，今日の視点で見れば，その機能はとくに優れたものではありません。しかし，今もなおソフトは続々と増え続けていて，なかでもゲーム・ソフトの質の高さは日本のパソコン・ゲームの比ではありません。

なぜ，こんな古ぼけたマシンの上で，そんなにすばらしいソフトが動くのでしょうか。それは前に述べたように，ソフトウェアがマシンの性能を100％発揮できるよう設計されているからなのです。もっとも，確かな技術力と豊かな創造力がそれを支えているという背景がものを言ってるわけですが……。

ＡＰＰＬＥのゲームのすばらしさは，マイコン雑誌でもよく話題にされますが，こうしたソフトを見るにつけ，日本とアメリカの差を思い知らされる気がします。

さてあまり感心してばかりいないで，実際にどうすればマシンの性能を100％発揮できるゲームをつくることができるかを考えてみることにしましょう。

1. マシン語によるゲームの開発

マイコン雑誌やソフト・ハウスのプログラム・コンテストがさかんですが，ある審査員は『全部BASICで書かれたリアルタイム・ゲームなんて見る気がしない。見ても想像できる範囲の作品しかない』と断言しています。

これはBASICでゲームを作ろうと思っている人にはショックかもしれません。しかし，現実的にはかなり的を得た発言だと思います。現にパソコン・ゲームのヒットチャートを見ても，上位のプログラムはマシン語で記述されたものばかり，仮にBASICで書かれたものがあっても主要部分にマシン語を利用している場合が多いのです。

プログラミングに時間を要し，デバッグが大変なマシン語がゲームの記述にこれだけ利用されるのはなぜでしょうか。それはインタープリタであるBASICでプログラムを記述した場合，CPUの能力の20～30%ぐらいしか発揮させることができないからなのです。コンパイラを使うという方法もありますが，実行時，プログラムに関する著作権等の問題が残ります。

そこで，マシンの能力を最大限に発揮させるということを考えた場合，どうしてもマシン語を使わなければならないという結論になってくるのです。

このようにマシン語を利用することで，具体的には次のような効果を期待することができるでしょう。

①速度の向上

ゲームを記述するためにマシン語を利用する最大の理由は，プログラムの実行速度が非常に速いということにあります。BASICと比較した場合，その処理速度は数倍から数100倍にもなり，この点でマシン語は明らかにBASICに勝れています。

とくにリアルタイム・ゲームではスピードが生命ですから，まずマシン語で記述されていることが大前提です。またそれ以外のタイプのゲームであっても，処理速度が速ければプログラムにそれだけ余裕ができるので，ゲームの内容により広がりを持たせることができます。

②きめの細かいプログラミングを可能にする

最近のBASICには豊富なステートメントが用意されていて，画面上に線を引いたり，音楽を演奏させたりすることが簡単にできます。しかし，BASICの機能にないことをしようとすると途端に困ってしまうでしょう。

マシン語によるプログラミングは，手間がかかりますが，きめの細かい処理を行なうことが可能です。高速のグラフィック表示や割り込み処理などはマシン語のお家芸だといえます。

③使用メモリの節約

これはプログラムの内容にもよりますが，一般的にマシン語のプログラムはBASICのプログラムに比べて実行時の使用メモリが少くて済みます。したがって，同じ16Kバイトの大きさのゲームであっても，マシン語によるゲームはメモリに余裕があるため，より多くのキャラクタを登場させたり，多彩なストーリー展開を実現することが可能です。パソコンのメモリは8K～64Kバイト程度が普通ですから，これらのメモリを少しでも有効に利用しようとするならマシン語でプログラムを記述するか，さもなければ高速な外部メモリ（ディスク・ドライブなど）を使う必要があります。

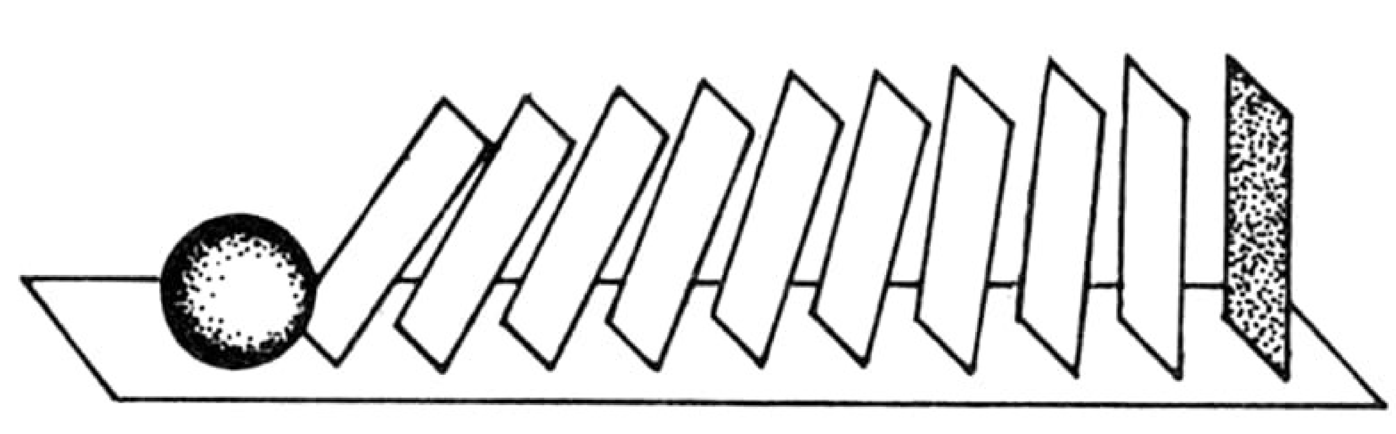
④プログラムの保護

マシン語によるプログラミングは手間がかかるという半面，内部が解析されにくいという利点もあります。たとえばBASICで書かれたアドベンチャーゲームには，LISTを見るとキーワードはおろか解法まで一瞬にしてわかってしまうようなものもあり，こうなるとアドベンチャーゲームとしての価値がなくなってしまいます。

その点マシン語のプログラムは解析されにくく，よほどのマニアでもなければ内容を知ることはできません。このようにプログラムの中身を知られることを防ぐということのほかに，プログラムをコピーから守るという点でもマシン語は有利です。

マシン語によるプログラミングは，ゲームをつくるうえでこのような数々の利点があります。もちろん，だからといってBASICのゲームがまったく駄目だというわけではありません。シミュレーション・ゲームのように数値計算を多用するゲームや文字処理を必要とするゲームでは，BASICの方が簡単にプログラムできるでしょう。またマシン語を使うほど速度を要求されないゲームもあります。

しかし，重要なことはBASICでできることはすべてマシン語で実現できるがその逆はできない，ということにあるのです。



2. ハードウェアをよく知る

ハードウェアの性能を最大限に発揮させるということは，ハードに対する十分な知識がなければできません。これは，別にハンダゴテをうまく使えるとか，抵抗のカラーコードが読めるとかいったことは違います。

つまるところ画面表示やキーの入力などのソフトウェアによるハードの使い方をよく理解しているかどうかにかかってきます。そのためにはパソコンに使われている主要なＬＳＩなどについて，十分，研究しておかなくてはなりません。

こうしてハードウェアの特性をよく見極めたうえで，それを生かすアイデアを見つければ，おのずと独創的で優れたゲームができあがるでしょう。

マシン語によるゲーム作り

4 MSXホームパソコンについて

『優れたソフトウェアはマシンの能力を100%引き出している。』ということを，これまで重ねて繰り返してきました。この見解にはほとんど異論がないと思われます。しかし，現実的な面から見た場合，こうしたマシンの能力を100%生かしたソフトにも問題がないわけではありません。

マシンの能力を最大限に生かすということは，そのマシン特有のハードを前提としたプログラムをつくる，ということであり，それは結果として異機種間での互換性を低下させることになります。たとえば，あるゲームが画面表示のためにパレット機能というのを使っていて，そのゲームを他機種に移植しようとしたとしましょう。もし移植しようとする機種にこのパレット機能がなければ移植は不可能になるか，プログラムの大幅な変更をしなければならなくなります。

どうせゲーム・ソフトなのだから特定の機種だけで動けばよい，という考え方もできるかもしれません。しかし，こうして対象を特定の機種に絞ってしまうことはそのゲームが商品になったとき，市場性という面で非常に不利になります。ではこの互換性を失わせずに，優れたソフトを組むにはどうしたらいいのでしょうか。

ＭＳＸは，こうした問題に悩むゲーム・プログラマーやソフト・ハウスに対する福音となるものです。

パソコンのユーザーにとって，メーカーや機種にかかわらずソフトが共通して使える，ということははかり知れないメリットがあります。しかし，異機種間で同じソフトが利用できるようにしたというだけならば，別にＭＳＸが最初というわけではなく，パソコンでもCP/M上で動くソフトはみんな共通して使うことができます。

ＭＳＸではこうしたソフトの互間性を保つために基本的なハードの部分までを標準化してしまった，というところに特徴があります。

したがってプログラマーは，ＭＳＸで規定されたハードの範囲内であれば，どのようなプログラムを書いても互換性についてまったく気にする必要がありません。このようにＭＳＸは，ソフトの利用者と開発者の相方に利益をもたらしてくれます。

ではいよいよこのあたりから本書の本題『ＭＳＸハードウェアの活用法』ということに話しを進めることにしましょう。

1. MSXの中核をなす3つのLSI

もし，今あなたの手許にＭＳＸマシンがあるのなら，ちょっとばかりケースを開けて，中身を覗いてみてください（といってもマシンを壊してしまうとメーカーは保障してくれませんので，念のため)。

首尾よくプリント基板がむき出しになったところでその上を調べると，40ピンのＬＳＩが４つ（機種によっては数が違うこともある）見つかります。

ひとつはＣＰＵのＺ80で，これについては何の説明もいらないでしょう。それから8255という番号の付いた同じ40ピンのＬＳＩがあるはずで，これは80系のＣＰＵでよく使われる汎用I/Oポートです。通称ＰＰＩといい，Ｚ80でマシン語のプログラムを組める人ならば一度ぐらいは使ったことがあるはず(？)です。

ところで残る２つのＬＳＩですが，実はこの２つとＣＰＵであるＺ80の３つのＬＳＩがＭＳＸのパソコンとしての性格を決定していると言えるものです。そのうちTMS-9918Aという番号の付いたものが一般にＶＤＰと呼ばれるＬＳＩで，ＭＳＸの画面表示機能のすべてをコントロールしています。それからもうひとつ残ったＡＹ-３-8910はＰＳＧと呼ばれるもので，ＭＳＸのサウンド機能をコントロールしています。

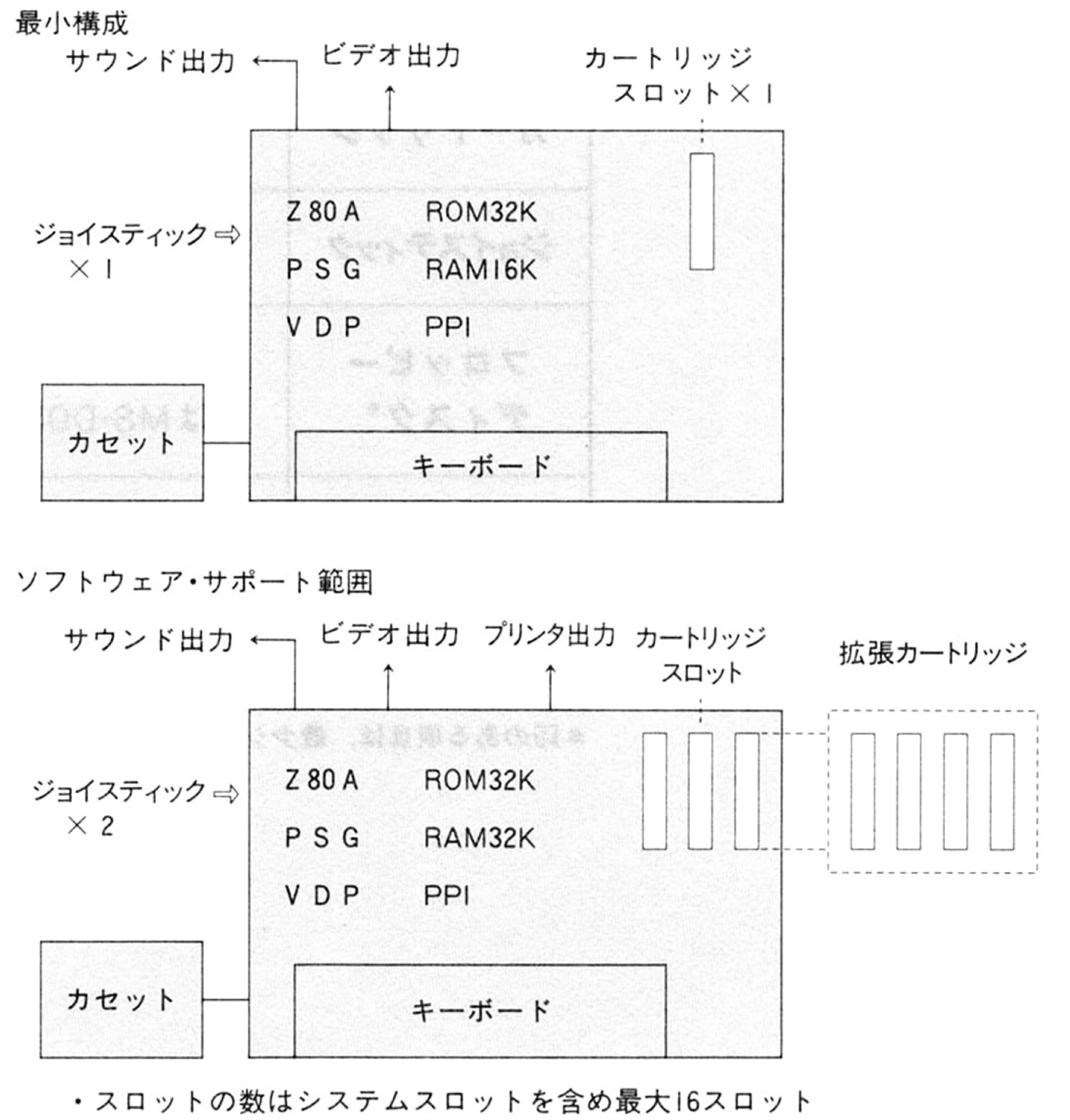
グラフィックスとサウンドは，それぞれＭＳＸの設計思想の中核となるものですから，この２つのＬＳＩの使い方をマスターすることができればＭＳＸシステムを使いこなしているといえます。そしてそこ

までいけば，あなたの目指すスタープログラマーへの道も目前でしょう。

本書では，次の第２章でまずＶＤＰについて取り上げ，第３章でＰＳＧについて解説します。

2. MSXの概略仕様

図1-1　システム構成



ＭＳＸの基本的な仕様は次のようになっています。

名　称	仕　　　様
CPU	Z 80 A 相当品
メモリ	ROM　32Kバイト(MSX-BASIC) RAM　16Kバイト以上
画面表示	VDP　TMS-9918A相当品 テキスト表示能力　40文字×24行 グラフィック　256×192ドット カラー　16色
サウンド機能	PSG　AY-3-8910相当品 ８オクターブ，３重和声出力

名称	仕様
CMT	FSK方式，1200/2400ボー
キーボード	英数，ひらがな，カタカナ，グラフィック記号対応，JIS配列/アイウエオ配列対応
ROM カートリッジ	I/Oバス，ゲーム・カートリッジ，拡張バスに対応するスロットを持つ
ジョイスティック	1または2*
フロッピーディスク*	3，3.5，5，8インチをサポート，フォーマットはMS-DOSに準拠
プリンタ*	8ビット・パラレル
漢字機能*	各社対応

*印のある項目は，最少システムにはなくてもよい

2章 MSXのグラフィック機能を使いこなす

VDP(TMS9918A)の使い方

MSXのグラフィック機能は，VDPと呼ばれるLSIでコントロールされています。ですから，このVDPの機能を知ることはMSXのグラフィックを使いこなすこと，ひいては最高にエキサイティングなゲームをつくることにつながります。

ここではVDPの機能の解説と具体的な応用法を考えていきます。

MSXのグラフィック機能を使いこなす

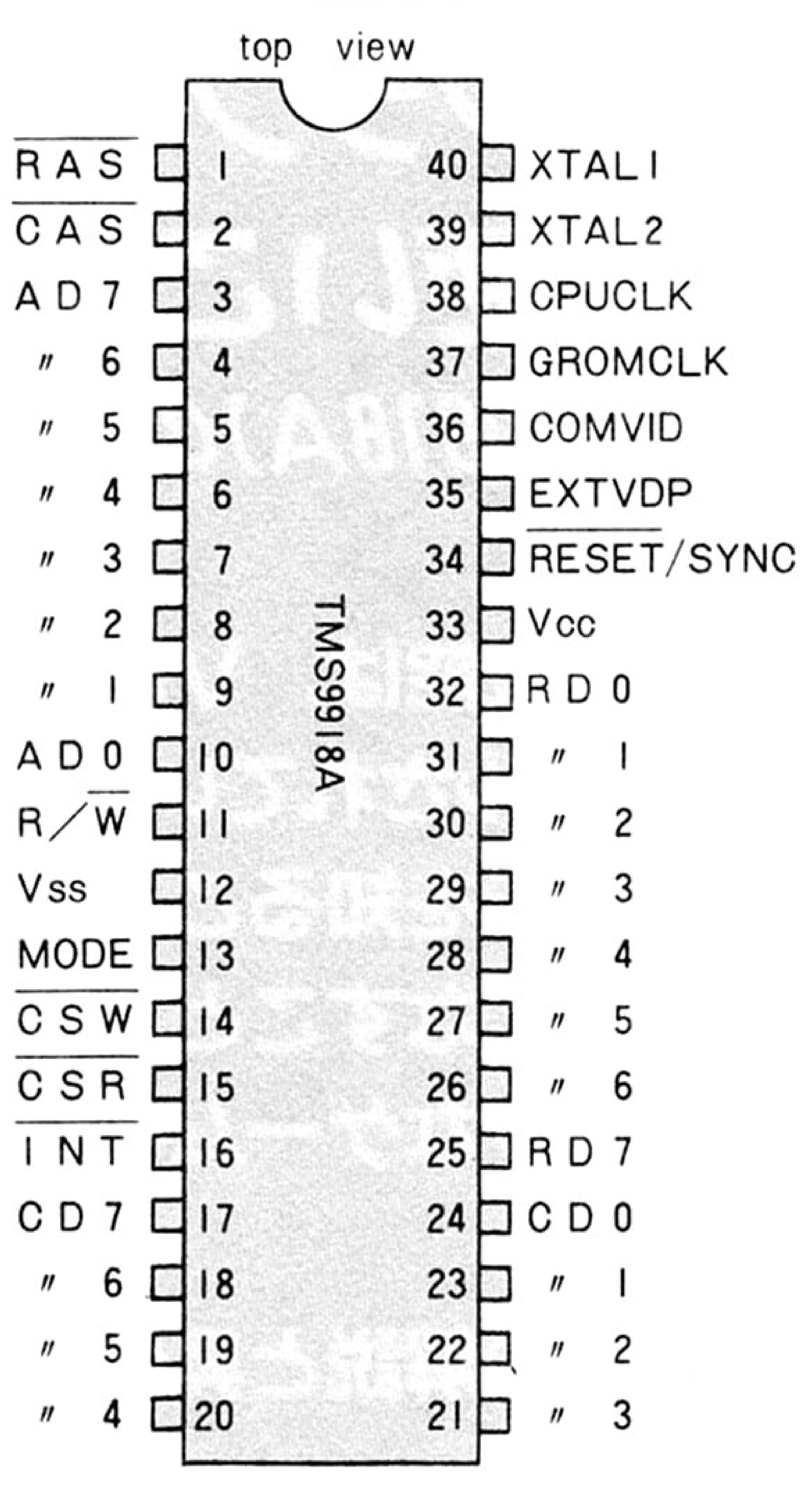
1 VDPについて

MSX規格では，画面表示用のLSIとしてTMS9918Aおよびその相当品を使用する，ということが決められています。このLSIがいわゆるVDP (Video Disply Processor) で，文字の表示やパターンの表示などパソコンに必要なグラフィック機能を1つのチップの上に組み込んだものです。

ワンチップながらその機能は非常に高く，かつてならこれだけのグラフィック機能を実現しようとすれば，1枚の基板が必要になったほどです。したがって，その集積度の高さは下手なCPUなどとは較べものになりません。

一方機能的な面からこのVDPを見ると，スプライト機能(後述)に代表されるように，ゲームをすることを主目的に設計されていることがわかります。すなわちVDPは単なるグラフィック・コントローラではなく，アミューズメント指向のグラフィック・プロセッサであるといえます。

図2-1　VDPピン配置図



Vss	GND
Vcc	5V
$\overline{RAS}$	VRAMの$\overline{RAS}$へ接続
$\overline{CAS}$	VRAMの$\overline{CAS}$へ接続
R/W	VRAMの$\overline{WE}$へ接続
MODE	モード
$\overline{CSW}$	チップセレクト・ライト
$\overline{CSR}$	チップセレクト・リード
$\overline{INT}$	CPUへのインタラプト
$\overline{RESET/SYNC}$	リセット
EXTVDP	外部ビデオ入力，MSXでは使用しない
COMVID	コンポジット・ビデオ信号出力
GROMCLK	接続しない
CPUCLK	XTALの1/3の周波数出力 (MSXではCPUのクロックとして使用，3.579545MHz)
AD7～AD0	VRAMのアドレス線へ接続
CD7～CD0	データ・バスに接続(CD0がMSB)
RD7～RD0	VRAMのODと接続
XTAL2, XTAL1	クロック入力 (MSXでは10.738635MHz)

ところでVDPにはこのTMS9918Aの他にTMS9928AというLSIがあり，MSXの中にはこちらの方を搭載している製品もあります。両者の違いは出力信号がビデオ信号であるかRGB信号であるか，ということですからソフト的には同じものと考えて差し支えありません。

MSXのグラフィック機能を使いこなす

2 VDPの特徴

前にも述べたようにVDPは非常に機能の高いLSIで，数々の特徴を持っています。こうした特徴を簡単にまとめると次のようになります。

①256×192ドットのグラフィック画面。

②スプライト機能を持つ。

③16色のカラー指定が行える(**表2-1**)。

④CPUとは独立したVRAM(MSXでは16Kバイトの大きさ)を持つ。

⑤4種類の表示モードを選択することができる。

⑥1画面の走査毎(1/60秒ごと)にCPUに対し割り込みをかけることができる。

⑦NTSCコンポジット信号を直接出力する（TMS9918Aの場合，TMS9928はRGB信号出力)。

⑧外部ビデオ信号との同期を取ることが可能（TMS9918Aのみ)。

表2-1 カラーコード表

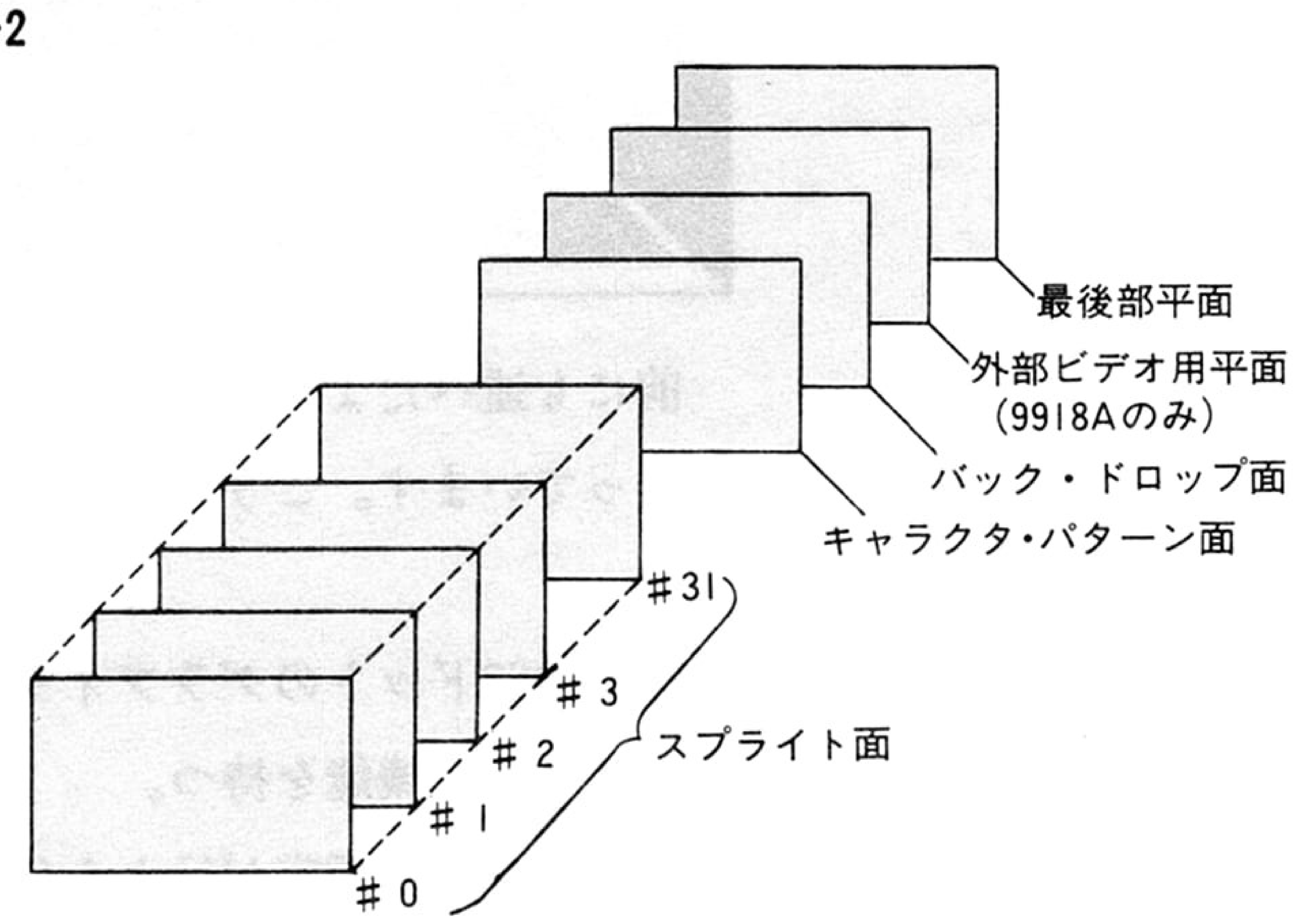
カラーコード	カラー	カラーコード	カラー
0	透明	8	赤
1	黒	9	淡い赤
2	緑	A	濃い黄
3	淡い緑	B	淡い黄
4	濃い青	C	濃い緑
5	淡い青	D	マジェンタ
6	濃い赤	E	灰色
7	シアン	F	白

こうした特徴の中で特に目を引くのは，やはりスプライト機能を持つということでしょう。スプライトといっても別に清涼飲料水のことではなく，一口でいえば移動キャラクタの表示を手軽に処理できるようにした機能のことをいいます。

そこでこのスプライトについて理解してもらうために，ＶＤＰの表示面の構成について解説することにしましょう。

ＶＤＰの表示面は丁度アニメのセル画のように何枚もの画面を重ね合わせた構造をしています(図2-2)。画面数は全部で36枚あり，各面は表示する内容により５種類に分けることができます。

図2-2



●最後部平面

ＶＤＰの画面の一番後ろにある面で，色は常に黒です。この最後部平面と次の外部ビデオ用平面は，通常使用されることはありません。

●外部ビデオ用平面

TMS 9918Aと外部ビデオ信号とのスーパーインポーズ用に使用されます。

●バック・ドロップ面

ボーダーカラーを表示するためのものです。画面上の外枠の部分に当ります。

●キャラクタ・パターン

スプライト以外の文字，パターンをこの面に表示します。このキャラクタ・パターン面への表示方法には，グラフィックＩモード，グラフィックⅡモード，テキスト・モード，マルチカラーモードという４種類のモードがあり，それぞれ表示の仕方に特徴があります。

●スプライト面＃０～＃31

画面の手前から32枚分はスプライトを表示するために使用します。それぞれのスプライト面につき，１つのスプライトを定義することができます。またスプライト面には優先順位があり，番号の小さいものほど優先順位は高くなります。

さて，いまここで移動キャラクタ(**図2-3a**)をスプライト面の＃０に表示し，背景(**図2-3b**)をキャラクタ・パターン面に描いた場合を考えてみましょう。画面上には背景と移動キャラクタが重ね合わされて**図2-4** のように表示されます。キャラクタを移動させても，背景とキャラクタが別の面にあるので，背景が描かれているキャラクタ・パターン面には全く手をつけずに済ませることができます。また同じように，スプライト面に表示した２つのパターンどうしが重なった場合にも，優先順位の高いスプライト面が手前側に表示され，重なり合った部分の処理はＶＤＰが自動的に行ってくれます。このようにスプライトを使用すると画面に見かけ上の奥行きを持たせることができるのです。

いままではこうした画面どうしの重ね合わせをしたい場合，ソフトウェアで処理するしか方法がありませんでしたが，スプライト機能を持つＶＤＰでは，こうした処理を自動的に行ってくれるのでゲーム等の制作が非常に楽になります。スプライトがＶＤＰの目玉機能といわれる理由はここにあるのです。

図2-3　スプライト面とキャラクタ・パターン面

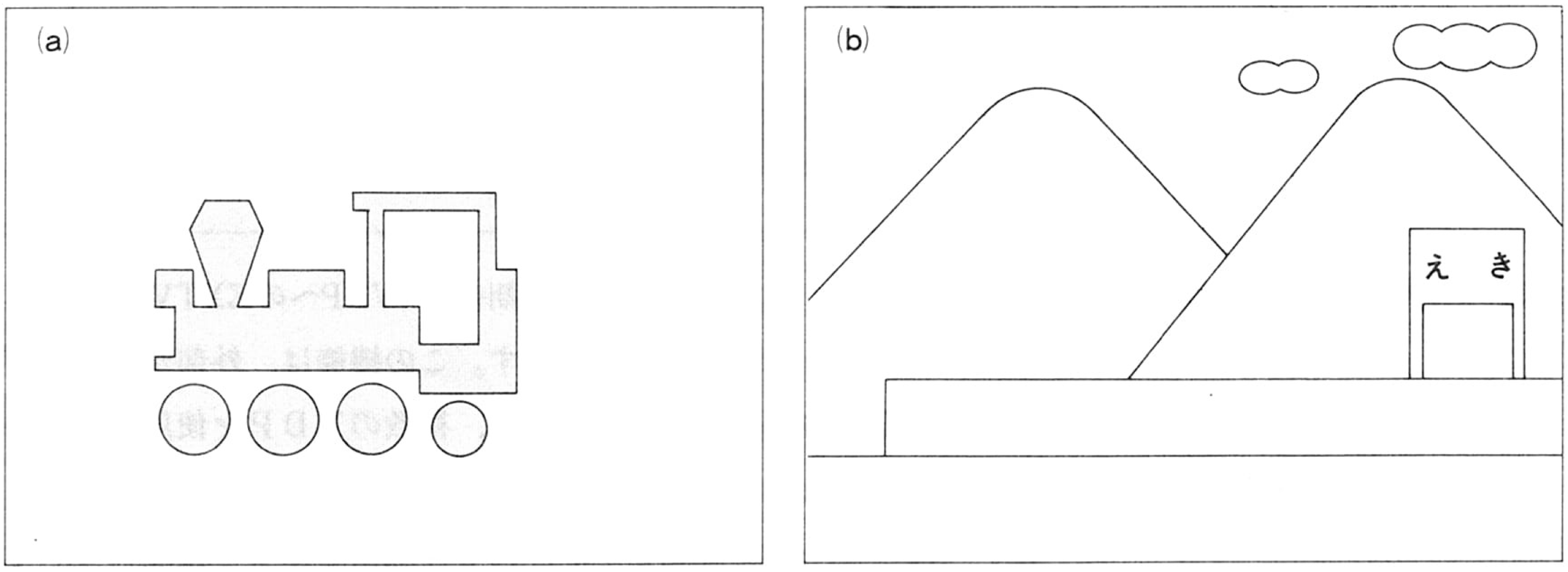
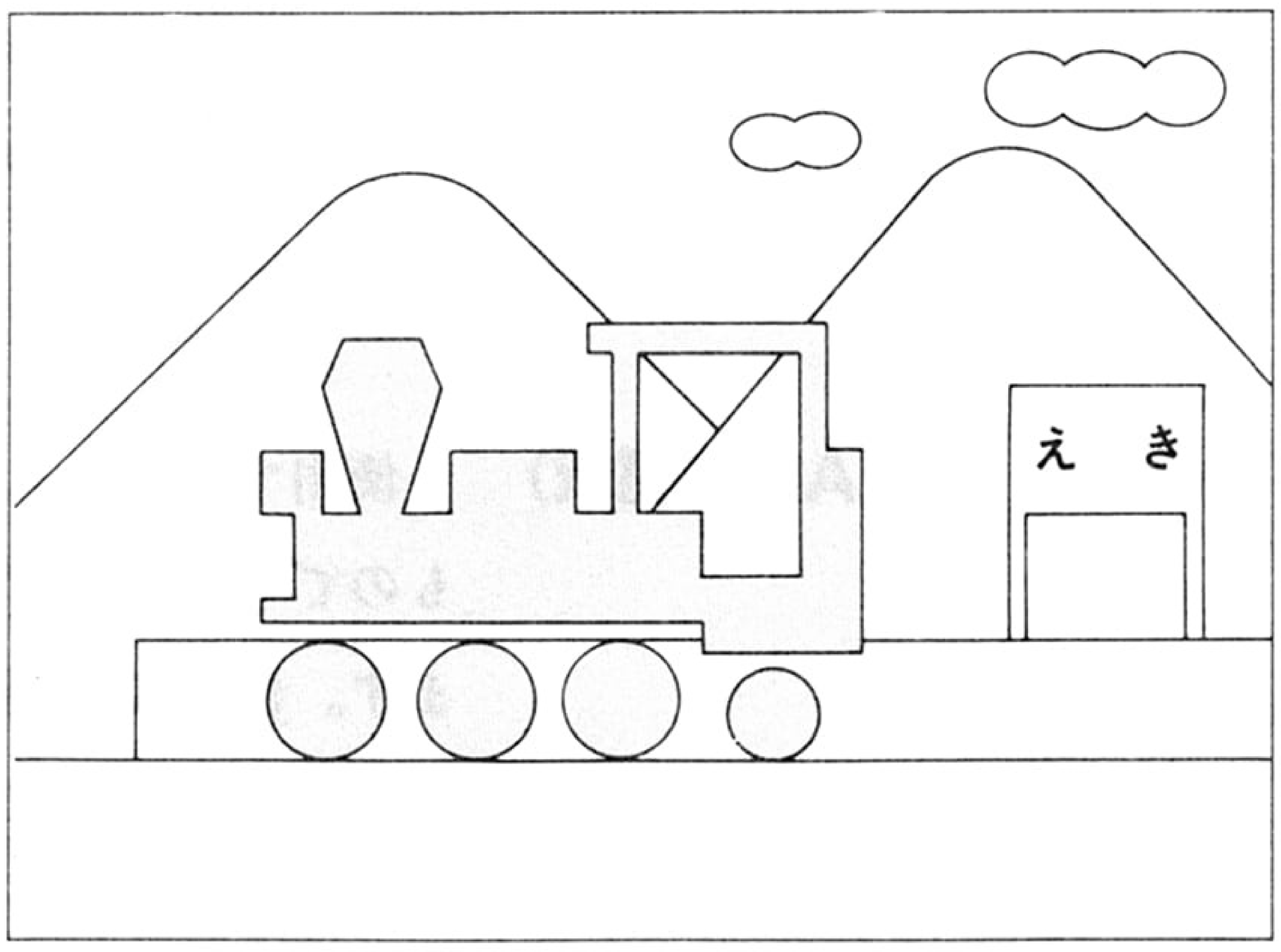


図2-4　合成された画面



MSXのグラフィック機能を使いこなす

3 VDPのレジスタ

VDPには8個の書き込み専用レジスタとVDPの状態を示す1個の読み出し専用レジスタがあります。MSXの画面表示は，これらレジスタの値とVRAM上のテーブルに書き込まれたデータによってコントロールされます。

1. レジスタ＃0，＃1

レジスタ＃0と＃1は，VDPの機能や表示モードを指定するためのものです。それぞれのレジスタの内容は**図2-5**のようになっていて，各ビットに1または0を書き込むことでVDPをコントロールすることができます。

図2-5　レジスタ ＃0，＃1

レジスタ＃0

7	6	5	4	3	2	1	0
0	0	0	0	0	0	M3	EV

レジスタ＃1

7	6	5	4	3	2	1	0
4/16K	BLANK	IE	M1	M2	0	SIZE	MAG

①EV（外部ビデオ信号）

EVビットを1にすると，外部同期信号（VDPへのEXTVDP入力）とVDPを同期させることができます。この機能は，外部からのビデオ入力とVDPの画面を合成させたり，複数のVDPを使用する場合を考えて設けられたものです。

たいへんユニークな機能ですが，MSX規格ではこの機能について触れていないため，一部の製品でしかサポートされていません。したがってMSX用のソフトを開発する場合は通常このビットを0にしておきます。

②4/16K（VRAMの選択）

使用するVRAMのサイズが4Kであるか16Kであるかを指定するものです。このビットに0を書き込むと4K，1で16Kの指定になります。MSXの場合はVRAMが16Kに決められているので，このビットは常に1になります。

③BLANK（画両のON/OFF）

BLANKに0を書き込むとVDPの出力が画面に表示されなくなります。再表示する場合は，新たに1を書き込みます。

④IE（インタラプト・イネーブル）

IEはCPUに対して割り込みを行うかどうかを指定するビットです。MSXではVDPからの割り込み（1/60秒ごと）を使って，TIME関数やキー入力などの処理を行っています。したがって，このIEを0（割り込みを禁止した状態）にすると，こうした機能が使えなくなります。

⑤M1，M2，M3

キャラクタ・パターン面への表示モードを指定するビットで，これらの値と表示モードの関係は**表2-2**のようになっています。

表2-2　モード選択

M1	M2	M3	表示モード
0	0	0	グラフィックI
0	0	1	グラフィックII
0	1	0	マルチカラー
1	0	0	テキスト

⑥SIZE（スプライトのサイズ）

VDPでは，スプライトのサイズに8×8ドット，または16×16ドットのどちらかを選択することができます。SIZEはこの指定を行うもので，0にしたとき8×8，1にしたとき16×16になります。

⑦MAG（スプライトの拡大）

このMAGビットを1にすると，スプライトの横方向，縦方向をそれぞれ2倍に拡大します。このMAGビットおよびSIZEビットの詳しい使い方についてはスプライトの項で解説します。

2. レジスタ#2～#6

レジスタ#2から#6までは，VRAMメモリ・マップを設定するためのものです。各種テーブルの先頭アドレスは，これらレジスタ#2から#6までに書き込まれる値によって決定されます。このテーブルについては，モードにより内容が異るので後述の各モードについての解説を参照してください。

①レジスタ#2（パターン名称テーブルの設定）

パターン名称テーブルの先頭番地の上位4ビットをこのレジスタのビット0からビット3までの4ビットにセットします（**図2-6**）。**図2-7**のメモリ・マップを例にとると，このレジスタの値は06Hになります。

図2-6　レジスタ　#2

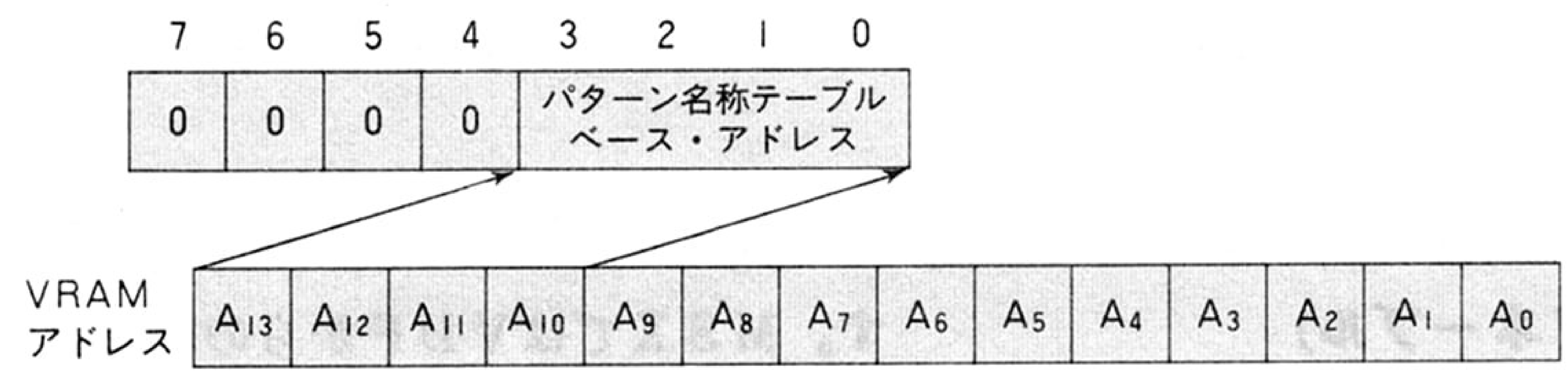
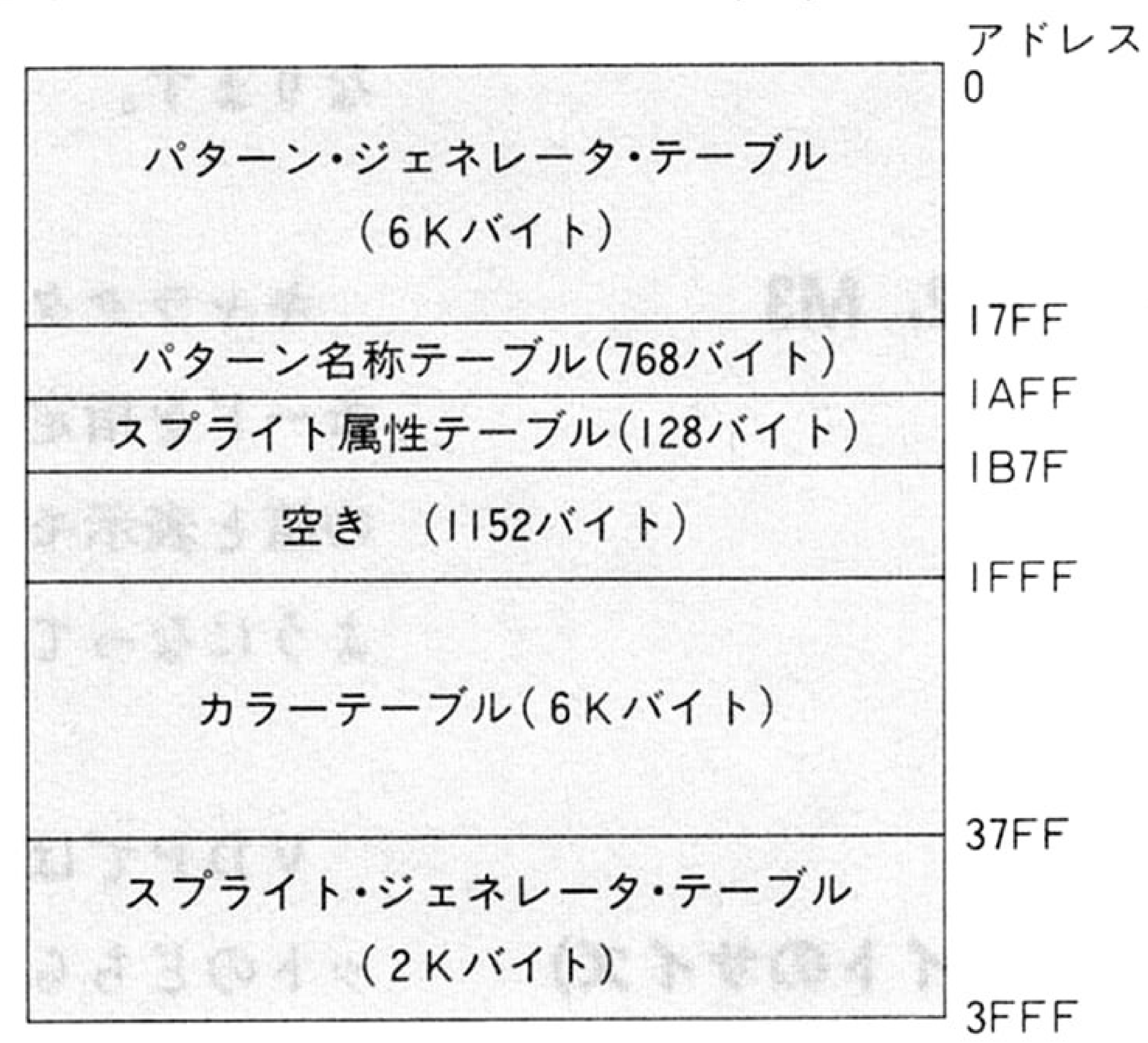


図2-7　VRAMのメモリ・マップ(例)



②レジスタ#3
(カラーテーブルの設定)

カラーテーブルの先頭番地の上位8ビットをこのレジスタにセットします(**図2-8**)。ただし，グラフィックIIモードでは，ビット7だけが有効で，他のビットはすべて1にしておかなくてはなりません(**図2-9**)。したがって**図2-7**の例ではこのレジスタの内容はFFHとなります。

図2-8　レジスタ　#3

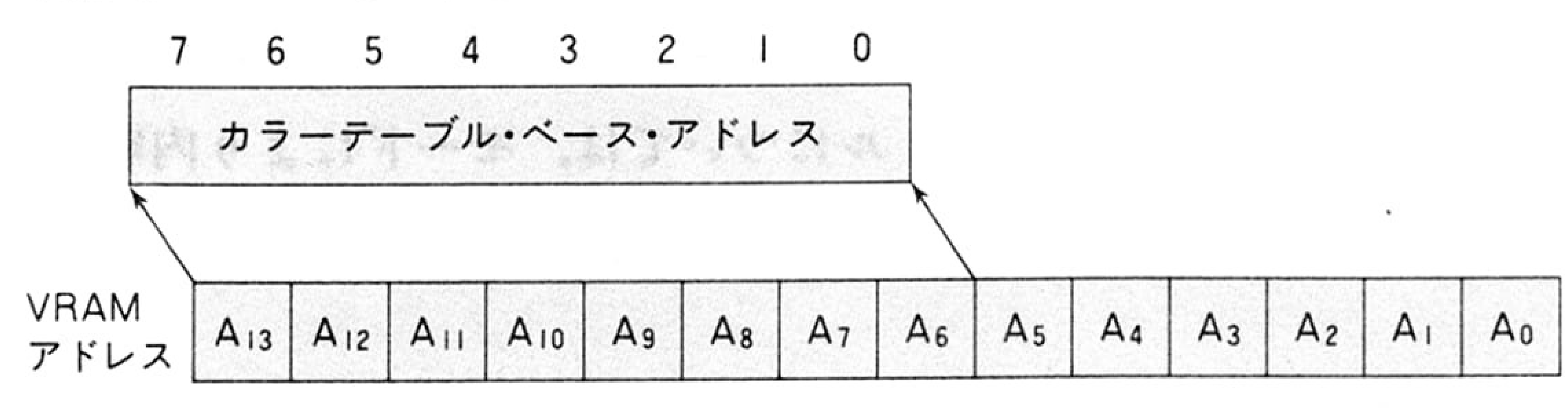
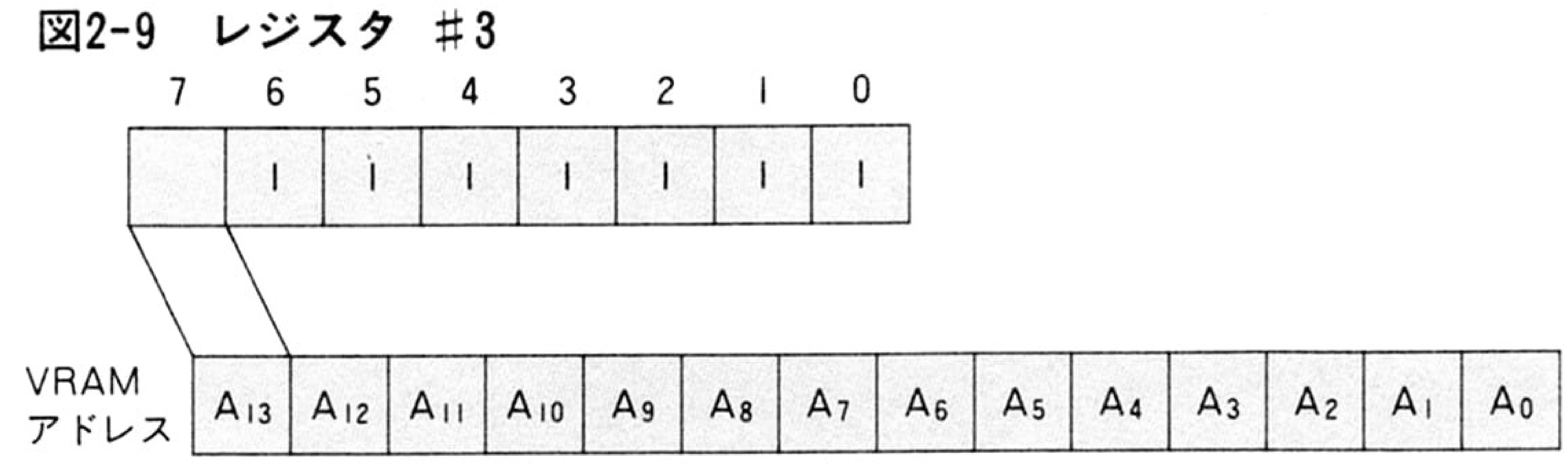


図2-9　レジスタ　#3



③レジスタ＃4(パターン・ジェネレータ・テーブルの設定)

パターン・ジェネレータ・テーブルの先頭番地の上位3ビットをこのレジスタのビット0からビット2までにセットします(**図2-10**)。ただし，グラフィックIIモードではビット2だけが有効で，ビット0とビット1は1にしておかなくてはなりません(**図2-11**)。したがって**図2-7**の場合では，このレジスタの内容は03Hになります。

図2-10　レジスタ　＃4

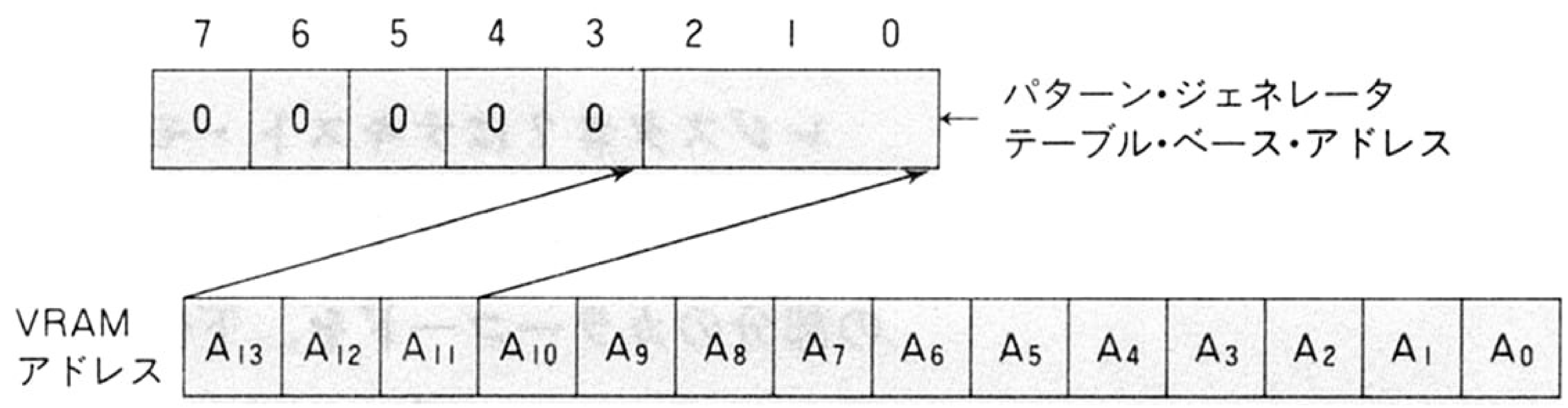
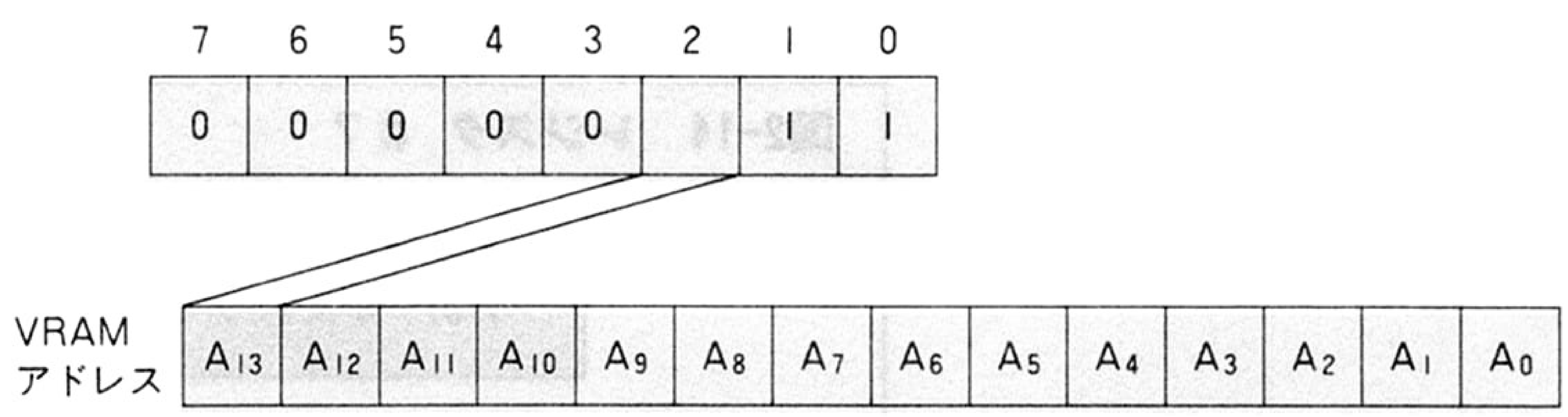


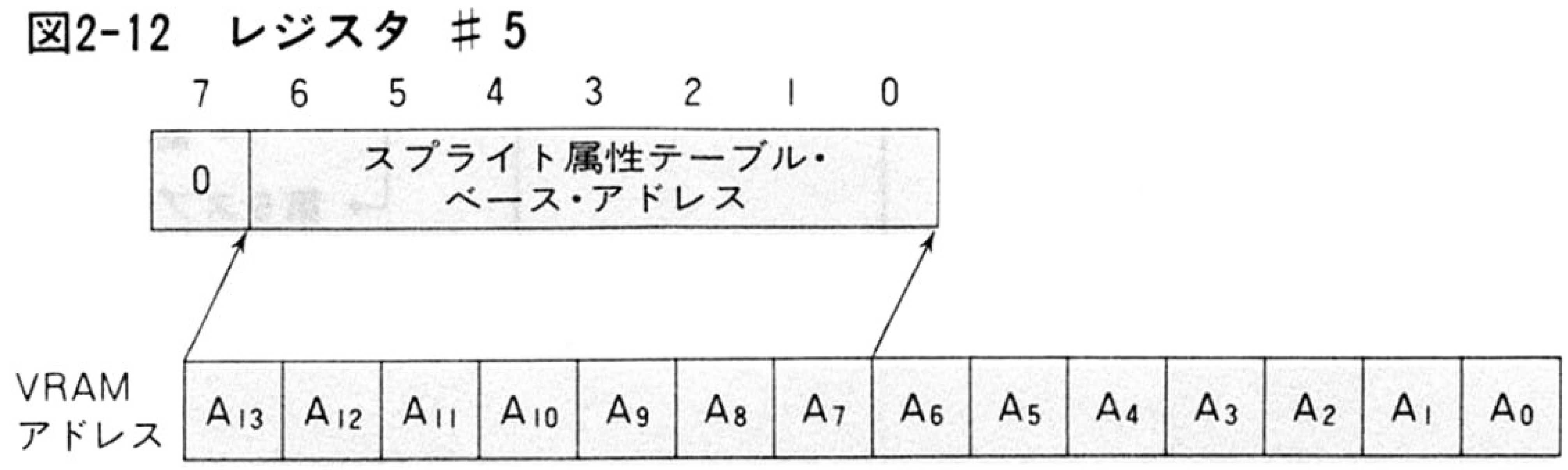
図2-11　レジスタ　＃4



④レジスタ＃5(スプライト属性テーブルの設定)

スプライト属性テーブルの先頭番地の上位7ビットをこのレジスタのビット0からビット6までにセットします(**図2-12**)。**図2-7**の場合このレジスタの内容は36Hになります。

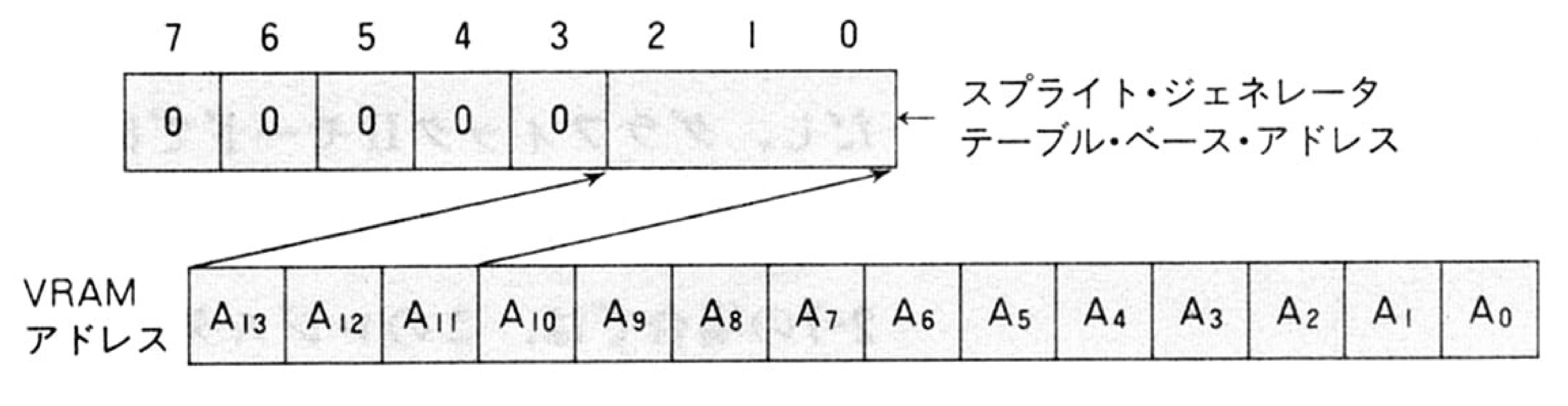
図2-12　レジスタ　＃5



⑤レジスタ＃6(スプライト・ジェネレータ・テーブルの設定)

スプライト・ジェネレータ・テーブルの先頭番地の上位3ビットをこのレジスタのビット0からビット2までにセットします(**図2-13**)。**図2-7**の場合，このレジスタの内容は07Hになります。

図2-13　レジスタ #6

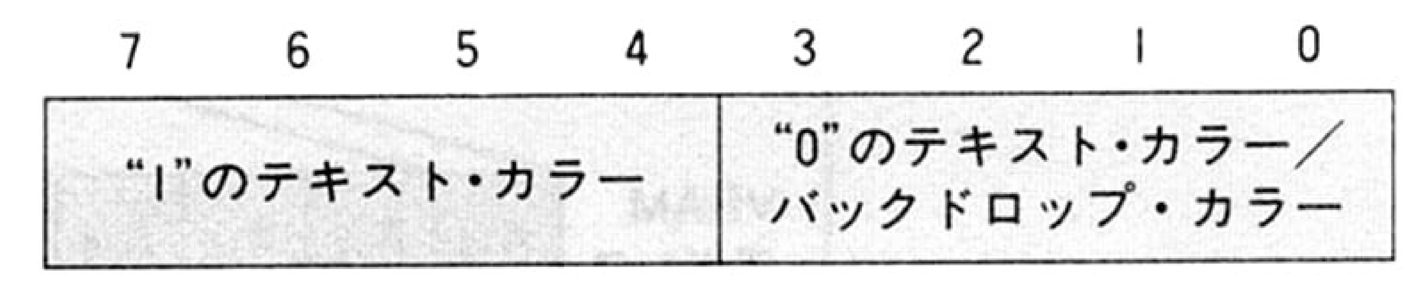


⑥レジスタ#7

レジスタ#7はテキスト・モード時のカラーを設定するものです。このレジスタの上位4ビットにキャラクタ・パターンのビットが"1"の部分のカラーコードを，下位4ビットには同じくビットが"0"の部分およびバック・ドロップのカラーコードをセットします(図2-14)。

VDPがテキスト・モード以外に設定してある場合，バック・ドロップのカラー指定をこのレジスタの下位4ビットで行います。

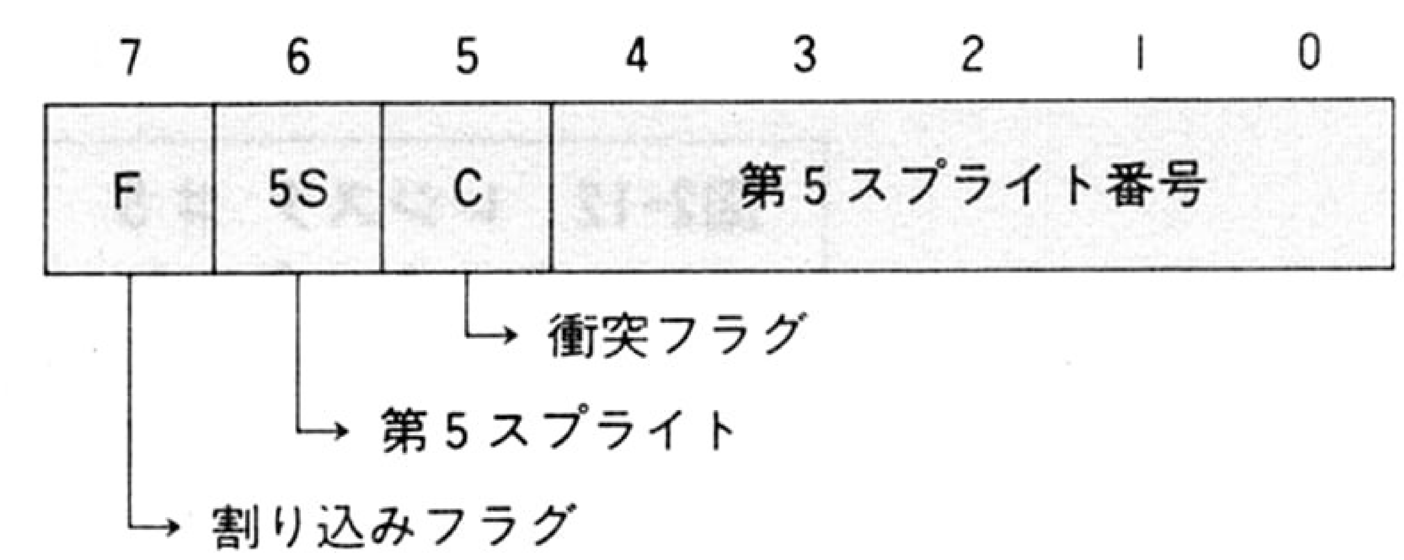
図2-14　レジスタ #7



3. ステータス・レジスタ

ステータス・レジスタは読み出し専用のレジスタで，このレジスタでVDPの状態を知ることができます。構成は図2-15のとおりです。

図2-15　ステータス・レジスタ



①F(割り込みフラグ)

レジスタ#1のIEビットが1にセットされている場合，1画面の画面走査終了毎（1/60秒ごと）にVDPのINT出力がアクティブになります。

②5S(第5スプライト番号)

5個以上のスプライトが同一水平ライン上に並び，かつ割り込みフラグ"F"が0の場合，5Sのビットが1にセットされます。

③C(衝突フラグ)

2個以上のスプライトのパターンが1の部分どうしが衝突(一致)した場合，1にセットされます。

MSXのグラフィック機能を使いこなす

4 表示モード

VDPには表2-3に示すような4つの表示モードがあり，これによってキャラクタ・パターン面への表示方法を決定します。それぞれの表示モードには機能などに特徴があり，目的に応じてユーザーが使い分ける必要があります。

表2-3　表示モード

表示モード	解像度	パターン・サイズ	パターン数	色	スプライト	表示画面
グラフィックI	256×192	8×8	256種類	16色	使用可	32×24
グラフィックII	256×192	8×8	768種類	16色	使用可	32×24
マルチカラー	64×48ブロック	1ブロック 4×4	―――	16色	使用可	32×24
テキスト	256×192	8×6	256種類	16色のうち2色	使用不可	40×24

1. グラフィックIモード

MSXを起動した直後の画面は，このグラフィックIモードになります。MSX-BASICではこのモードを文字の表示だけに利用していますが，文字以外のパターンを表示することもでき，スプライトも使えます。ただし，画面上の任意の場所に点を打つというようなことはできません。この場合は後述のグラフィックIIモードを利用することになります。

さてこのグラフィックIモードの特徴は，基本的に8×8ドットの大きさのパターンを32列×24行に表示するということです。このパターンは256種類まで定義でき，それぞれのパターンごとに色の指定が行えます。MSX-BASICでは，この256パターンすべてに表示文字を割り当てていますが，パターンを再定義して使うことができます。

グラフィックIモードで表示を行うときは，パターン・ジェネレータ・テーブル，パターン名称テーブル，カラーテーブルという3つのテーブルを使用します。ここでそれぞれの役割を簡単にまとめると，

- **●パターン・ジェネレータ・テーブル**
  8×8ドットのパターン・データを定義し，格納しておく領域。
- **●パターン名称テーブル**
  パターンの表示位置を指定する領域。
- **●カラーテーブル**
  表示するパターンの色を指定する領域。

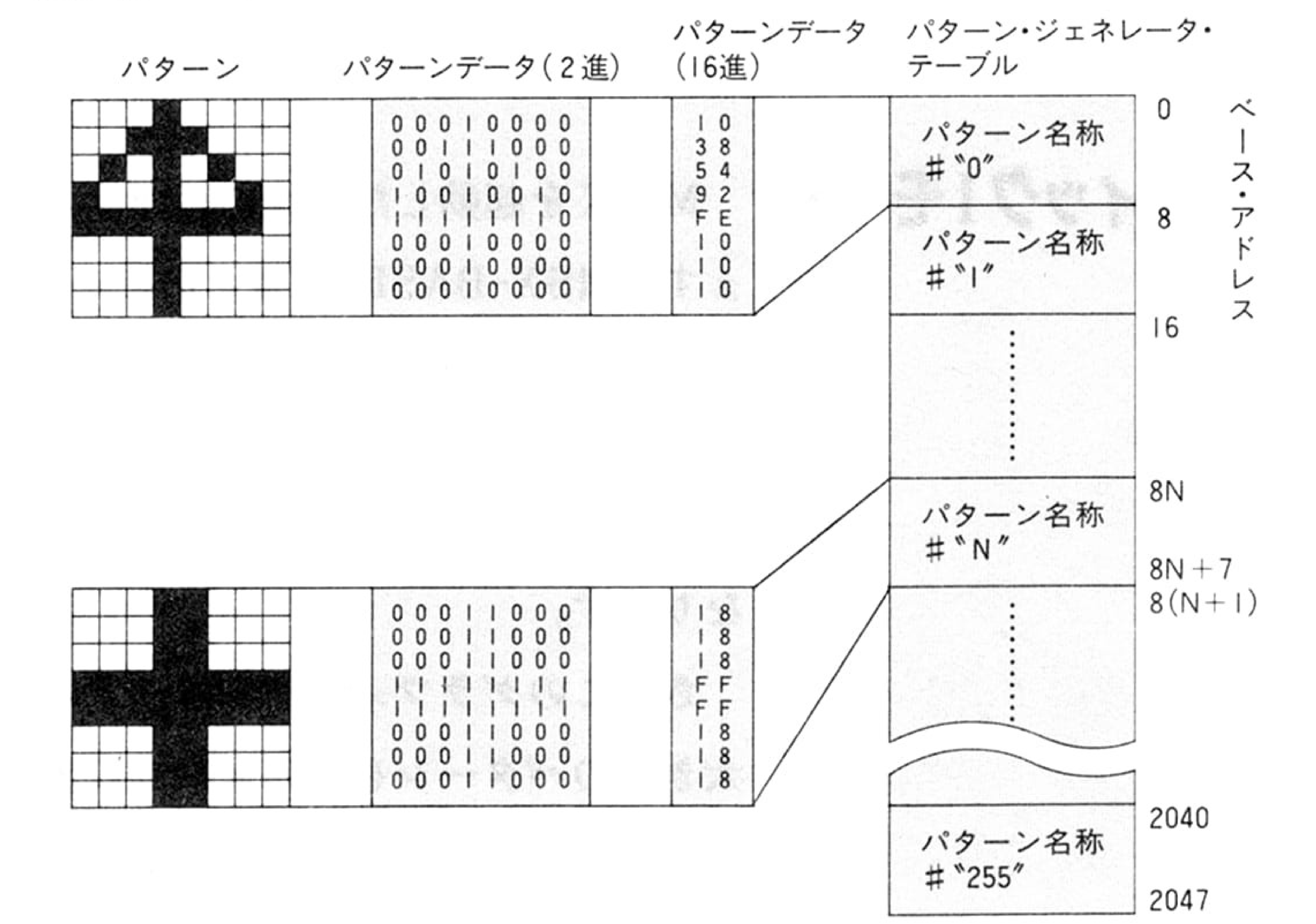
というようになります。ではさっそくこれらテーブルの定義方法を解説することにしましょう。

①パターン・ジェネレータ・テーブル

パターン・ジェネレータ・テーブルには，８×８ドットの大きさのパターンを256種類まで定義できます。それぞれのパターンは８バイトで構成され，**図2-15**のような形でテーブルに格納されます。テーブルの大きさは，８バイト×256＝2048バイトになります。

さてパターン・ジェネレータ・テーブルにパターンを定義すると，パターンを格納した位置によって自動的にパターン名称というものが決定されます。このパターン名称は**図2-15**に示すように，パターン・ジェネレータ・テーブルの先頭から，8バイト区切りで0, 1, 2……255という順に付けられ，以後このパターン名称がそのパターンを代表します。

図2-15　パターン・ジェネレータ・テーブル

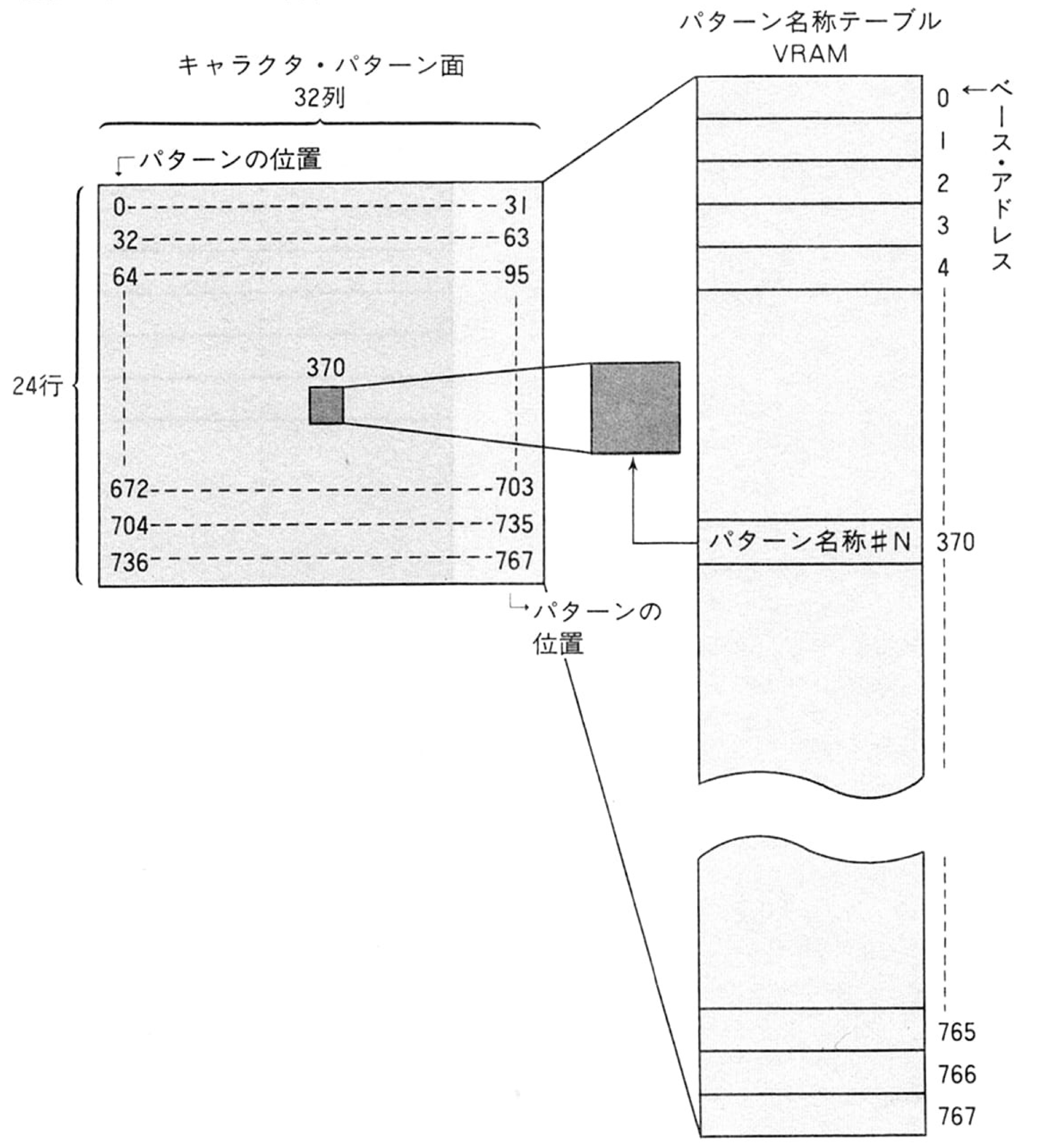


②パターン名称テーブル

パターン名称テーブルは，キャラクタ・パターン面と**図2-16**のような形で１対１に対応します。大きさは32列×24行で768バイトです。このパターン名称テーブルにパターン名称を書き込むと，キャラクタ・パターン面の対応する位置に，パターン名称によって指定されたパターンが表示されます。

ＶＤＰを使用していない一般のパソコンでは，文字表示用のＶＲＡＭに文字コードを直接書き込むと，そこに文字が表示されますが，パターン名称テーブルはこのＶＲＡＭに相当するものです。

図2-16　パターン名称テーブル

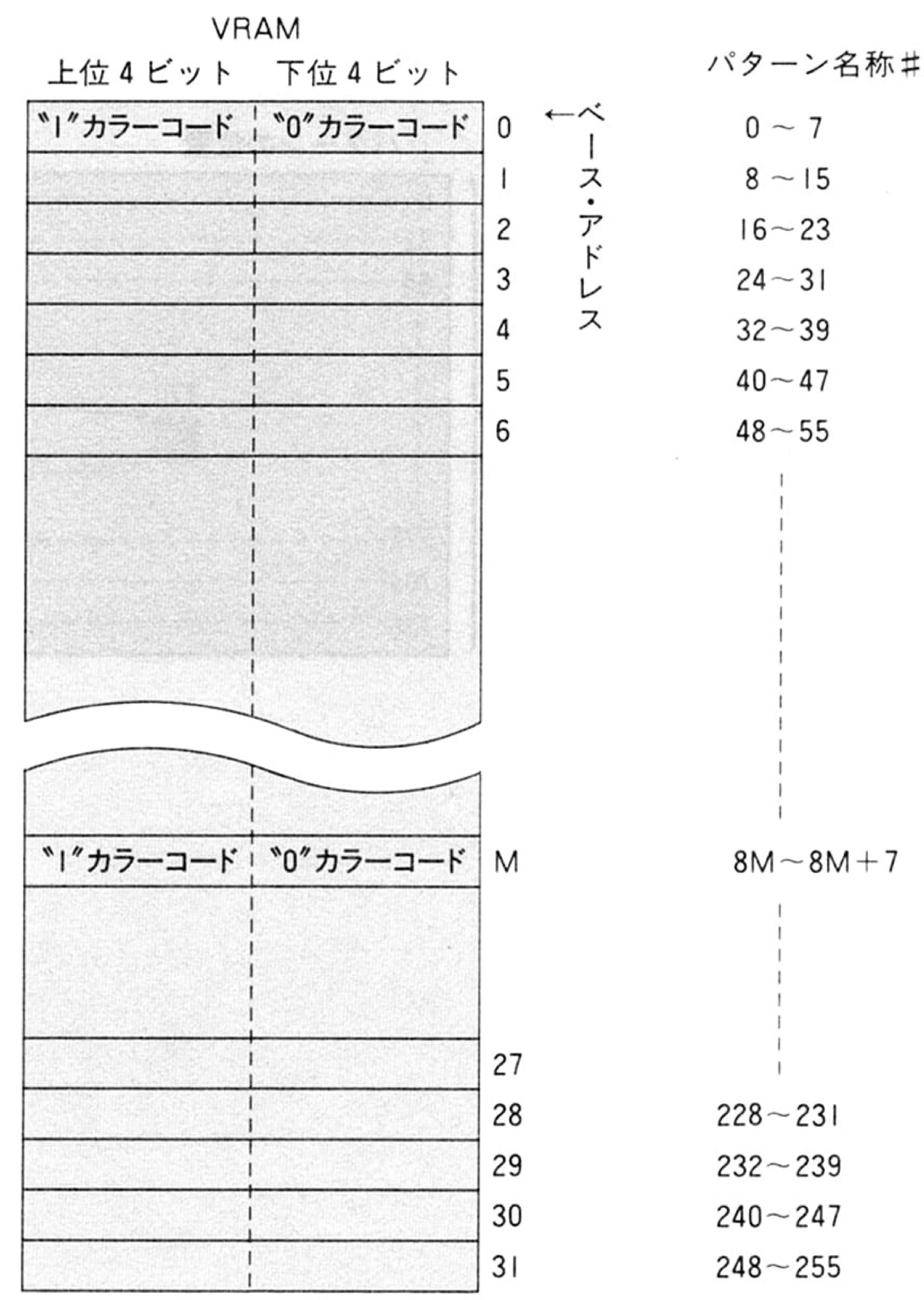


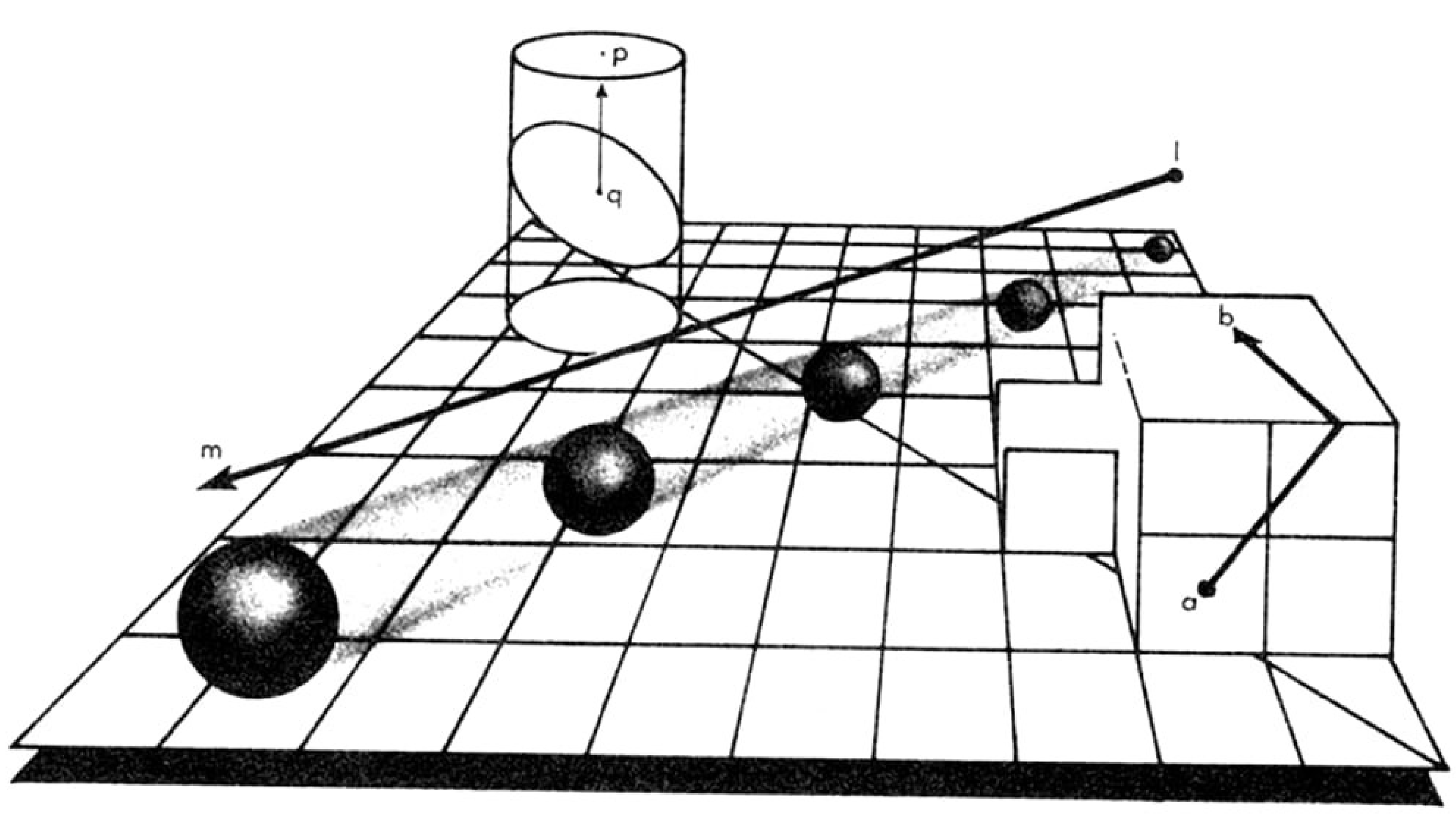
③カラーテーブル

パターン・ジェネレータ・テーブルに入れたパターンの色をこのカラーテーブルで指定します。この色指定は8つのパターンを組みにして行います。すなわち，パターン名称が0～7のパターンについて，パターンのビットが"1"の部分と"0"の部分のカラーコードをそれぞれ指定し，同じようにパターン名称が8～15のパターンについても2色を指定します。こうしたパターンとカラーテーブルの関係は**図2-17**のようになります。

カラーテーブルでは，各バイトの上位4ビットにパターンのビットのうち"1"の部分のカラーコードを，下位4ビットに"0"の部分のカラーコードを入れます。このカラーコードと色の対応については，前掲のカラーコード表(**表2-1**)をご覧ください。

図2-17　カラーテーブル



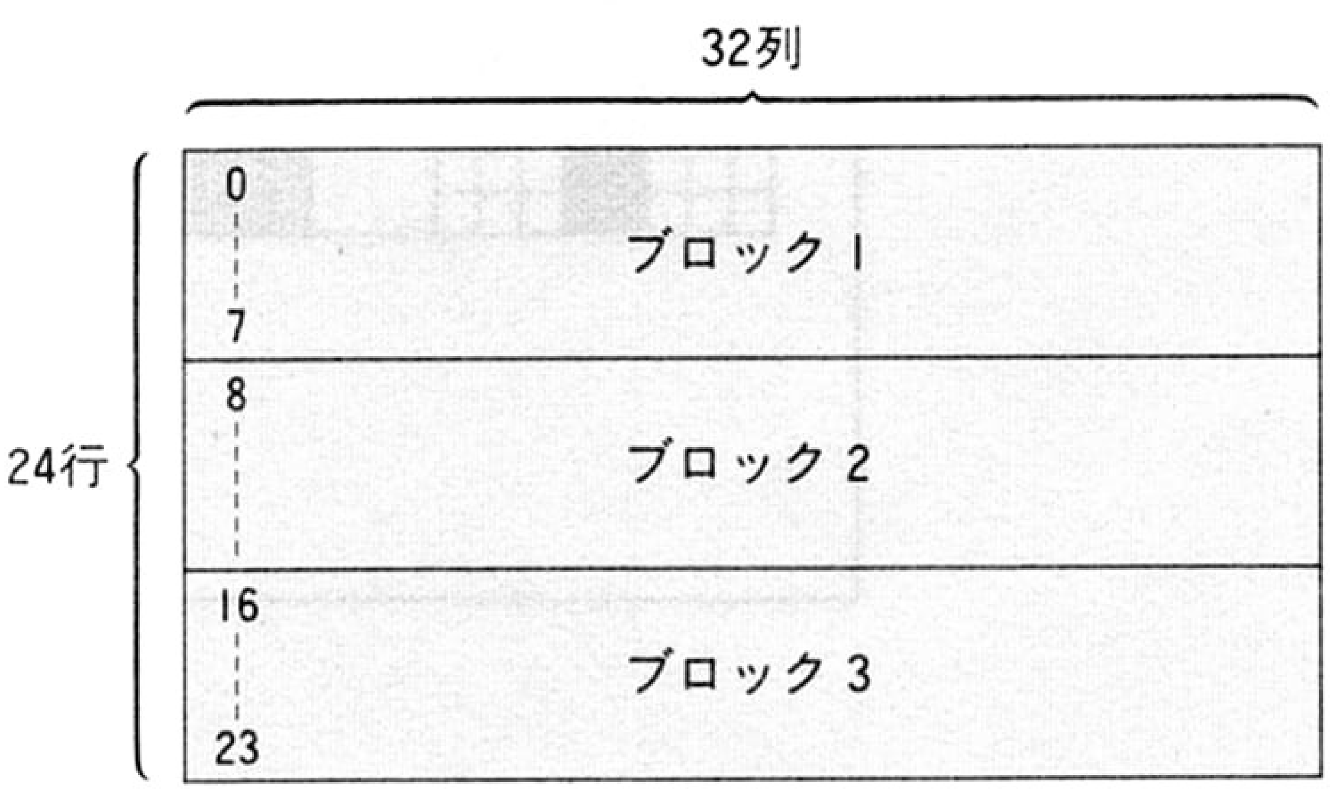


2. グラフィックIIモード

ゲームをつくるという立場からＶＤＰを見た場合，最も利用価値の高いのがこのグラフィックIIモードです。基本的にはグラフィックＩモードと同じですが，定義できるパターンの数が256種類から768種類に増えています。これにより，画面に点を打ったり線を引いたりするフルグラフィック機能が使えるようになります。

グラフィックIIモードでは，キャラクタ・パターン面を３つのブロックに分割して扱います(**図2-18**)。パターン・ジェネレータ・テーブル，パターン名称テーブル，それにカラーテーブルは，それぞれのブロックごとに別々に設定します。

図2-18　グラフィックIIモードでのキャラクタ・パターン２面



①パターン・ジェネレータ・テーブル

パターン・ジェネレータ・テーブルは，キャラクタ・パターン面と対応した３つのブロックで構成されます(**図2-19**)。

そしてそれぞれのブロックには，８×８ドットのパターンを256種類ずつ，計768種類のパターンを定義することができます(**図2-20**)。定義されたパターンには，テーブルに格納された順序によって，ブロックごとに０～255のパターン名称が与えられます。

このように扱えるパターンの数が増えたので，テーブルの大きさは，グラフィックＩモードの３倍の6144バイトが必要です。

図2-19　パターン・ジェネレータ・テーブルとキャラクタ・パターン面

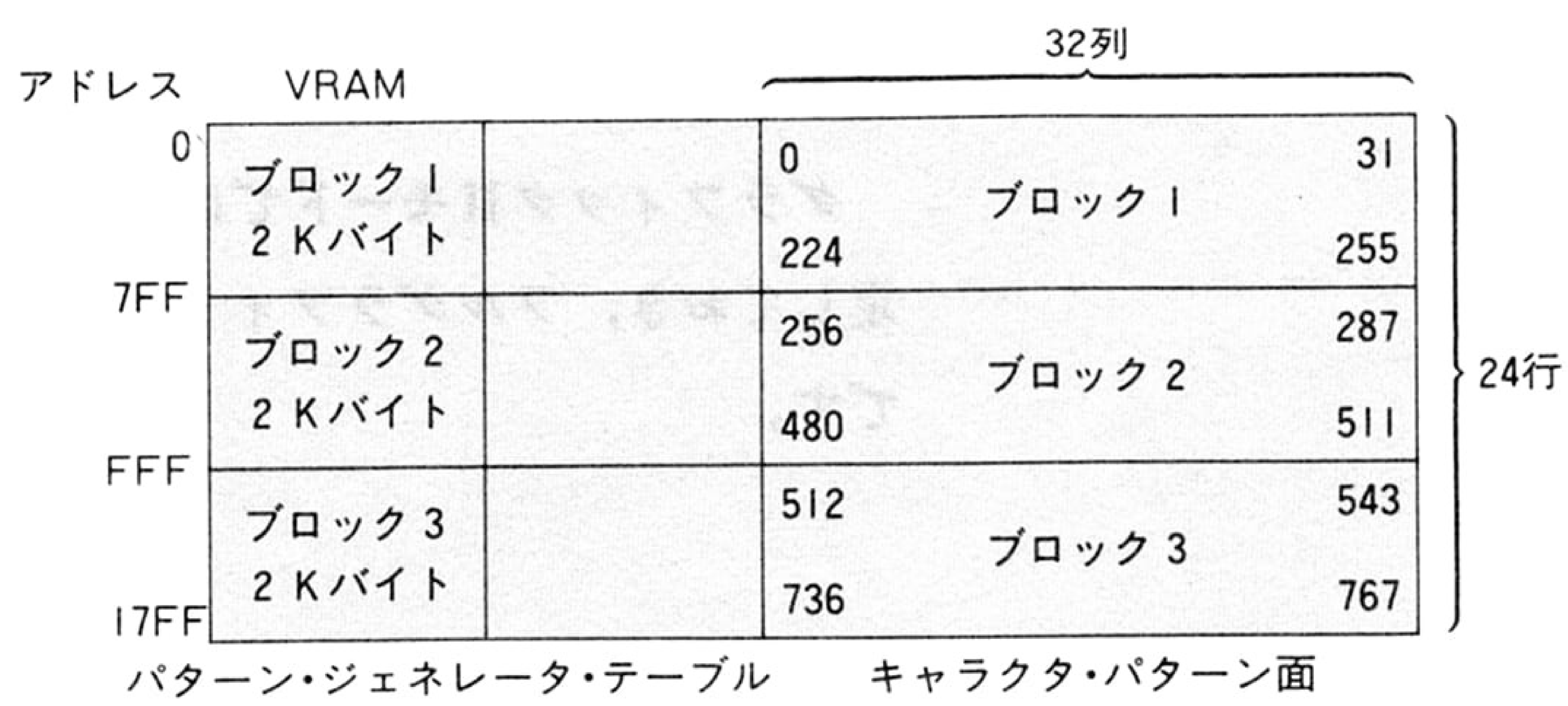
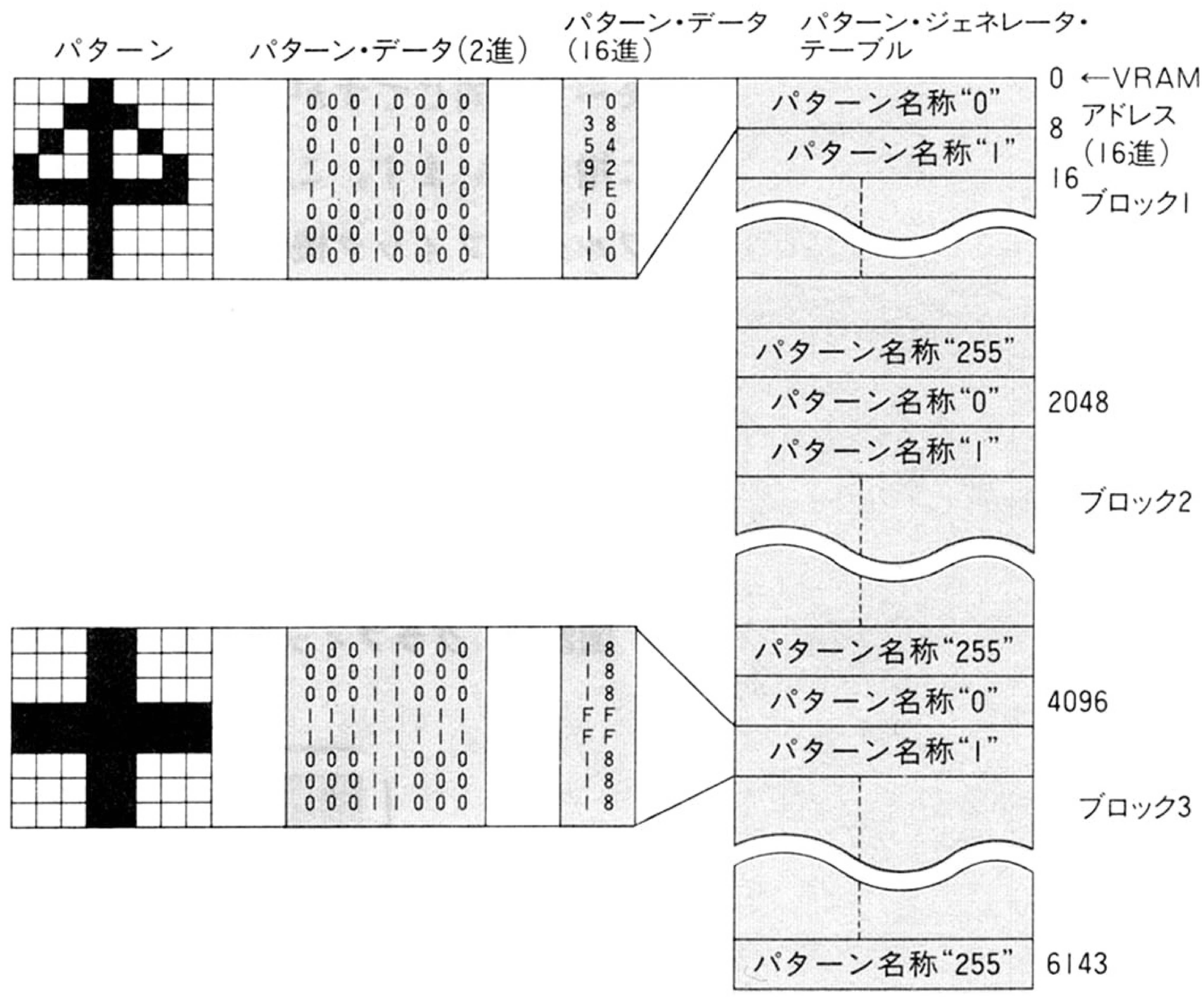


図2-20　パターン例とパターン・ジェネレータ・テーブル



④パターン名称テーブル

パターン名称テーブルもパターン・ジェネレータ・テーブルと同じく3つのブロックからなります。キャラクタ・パターン面とパターン名称テーブルの対応は，**図2-21**を参照してください。

このパターン名称テーブルのあるブロックにパターン名称を書き込むと，パターン・ジェネレータ・テーブルの対応するブロックから，パターンが取り出されて表示されます。あるブロックで定義されたパターンを他のブロックに表示することはできません。

さてここでパターン名称テーブルを**図2-22**のような内容にセットしてみることにします。こうするとパターン・ジェネレータ・テーブルとキャラクタ・パターン面が1対1に対応することになり，キャラクタ・ジェネレータ・テーブルを一般のグラフィックVRAMと同等に扱うことができます。

グラフィックIIモードでは，このようにパターン名称テーブルを固定しておき，フルグラフィック画面のような扱い方をするのが一般的です。

図2-21　キャラクタ・パターン面とパターン名称テーブル

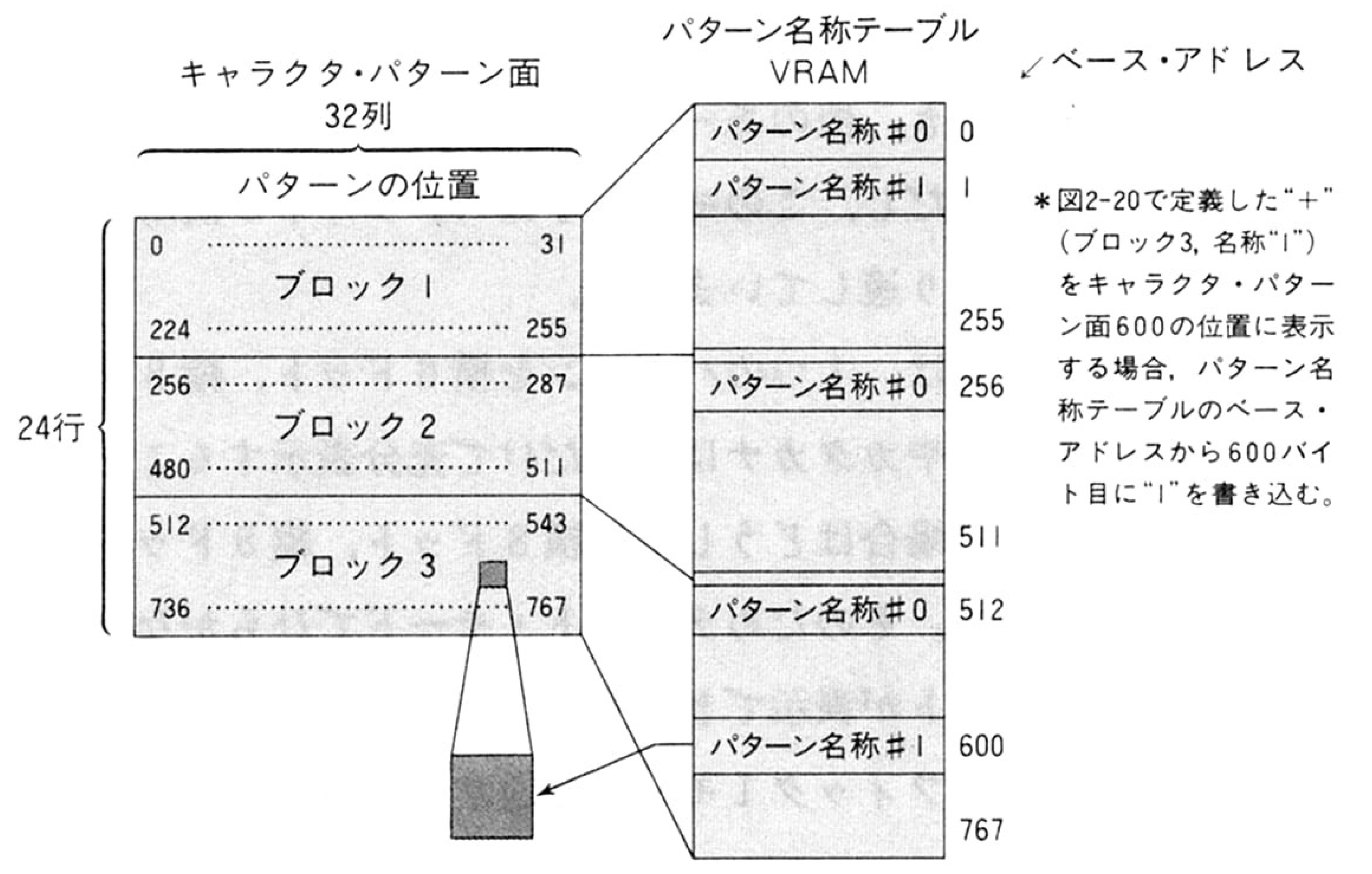
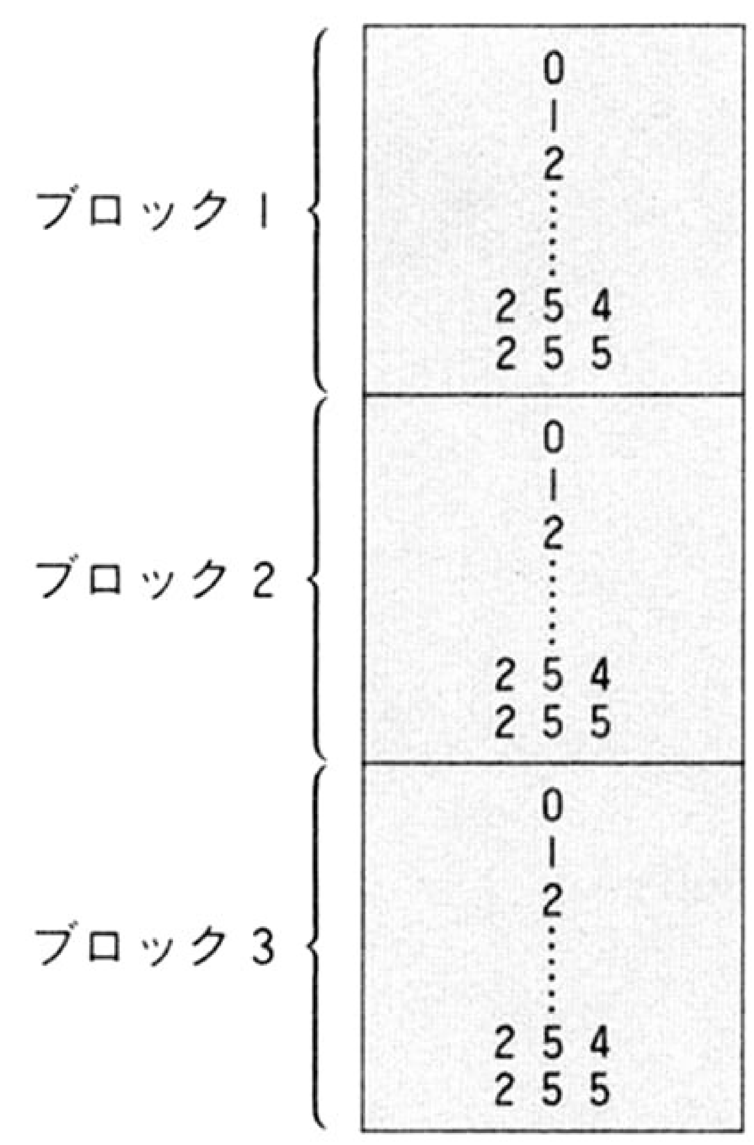


図2-22　パターン名称テーブルの設定

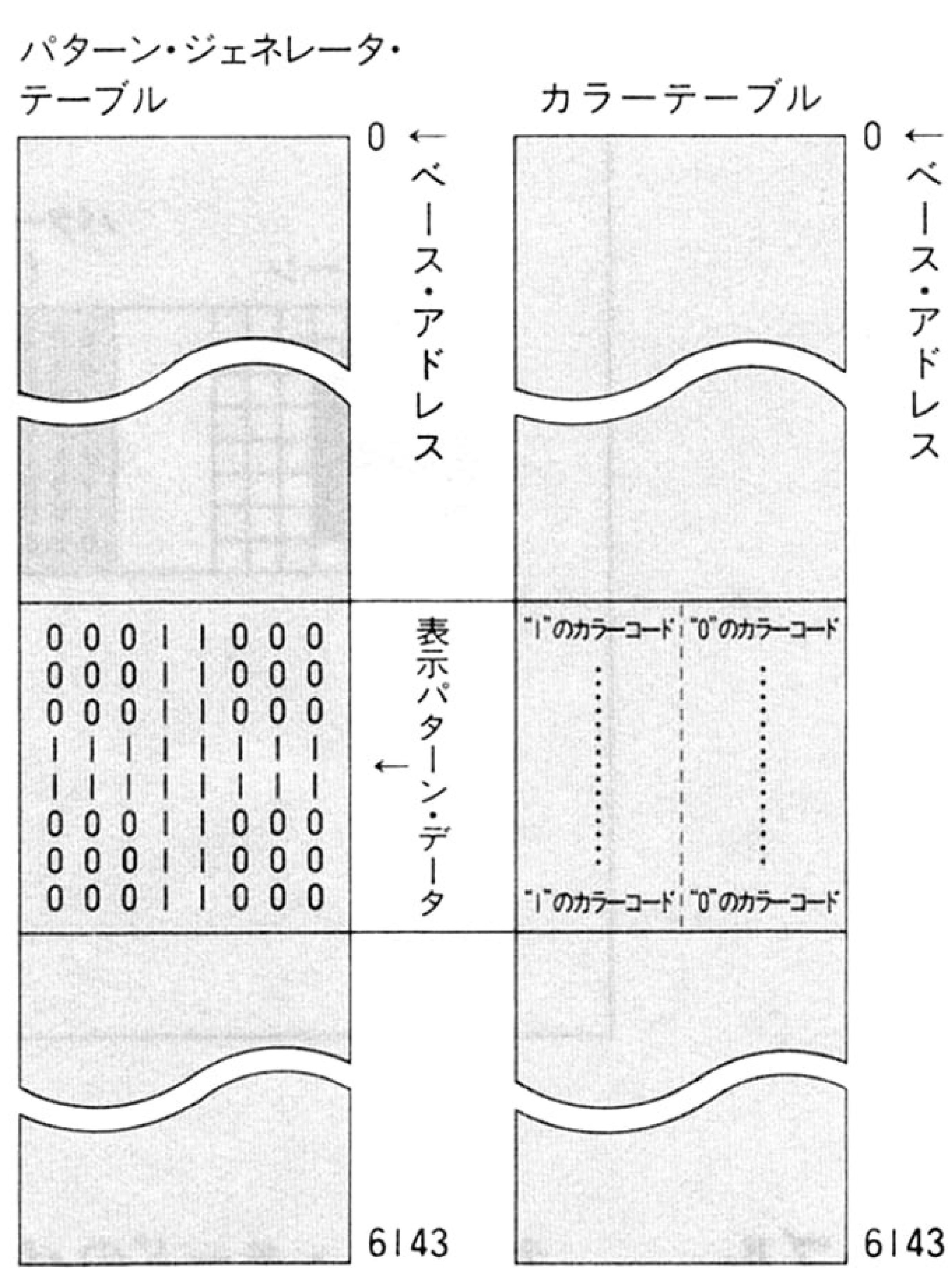


③カラーテーブル

グラフィックIIモードでは，1つのパターンについて1ラインごとにビットが"1"の部分と"0"の部分の2色を指定することができます（**図2-23**）。

カラーテーブルはパターン・ジェネレータ・テーブルと1対1に対応していて，上位4ビットにそのラインの"1"の部分のカラーコード，下位4ビットに"0"の部分のカラーコードを入れます。テーブルの大きさは，パターン・ジェネレータ・テーブルと同じ6144バイト必要です。

図2-23　表示パターンとカラーテーブル



3. テキスト・モード

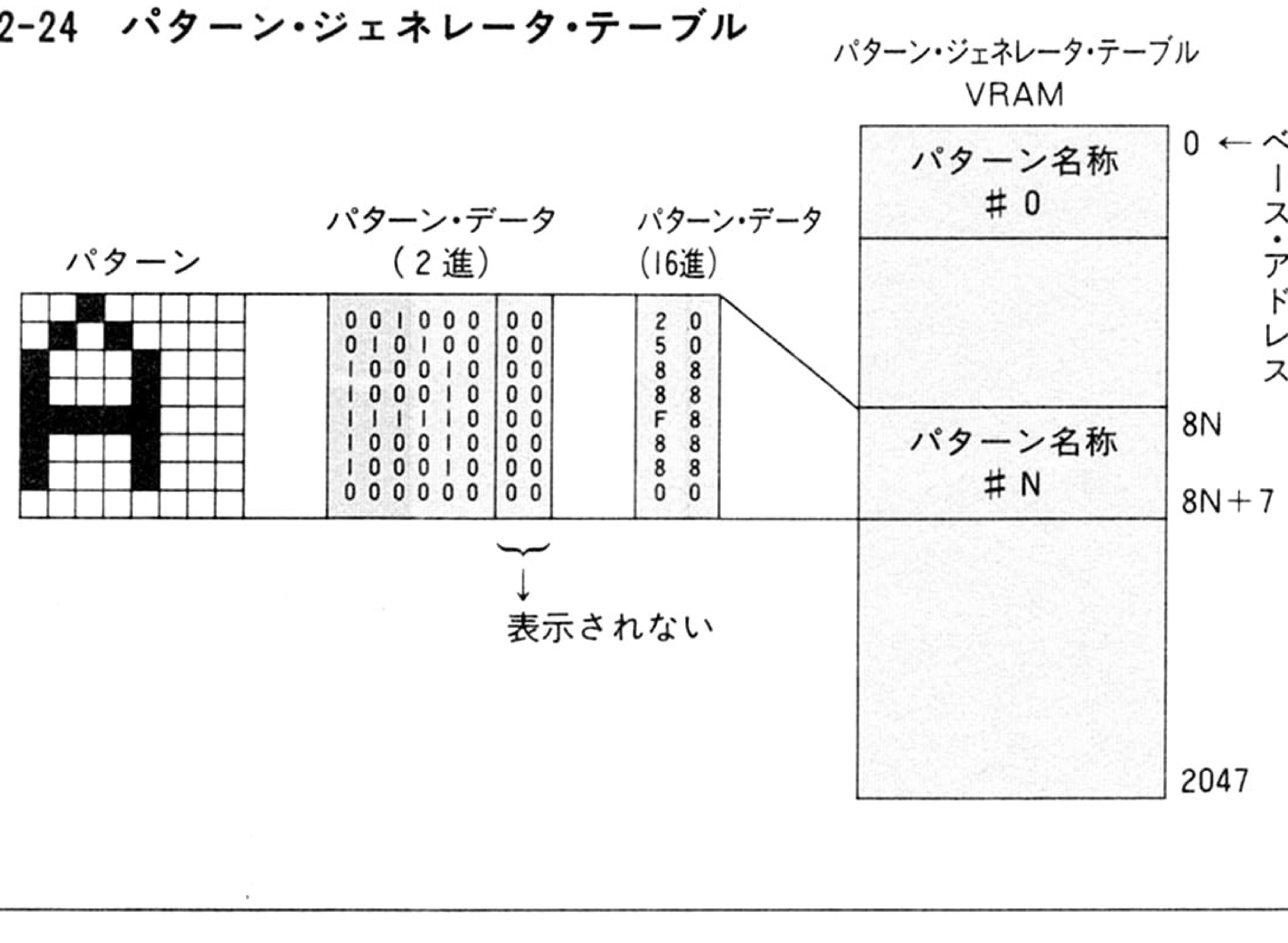
MSX-BASICでSCREEN 0を指定するとこのモードになります。もともとテキストの表示を目的にしたモードなので，横40×縦24文字の表示を行うことができ，他のモードよりも一画面に表示できる文字数が多くなります。ただし，このモードではスプライトが使用できず，図形を扱うのにもあまり適していません。

テキスト・モードでは，1つのパターンを横6ドット，縦8ドットで表わします。英数字やカタカナはこれだけで充分表示することができますが，ひらがなの場合はどうしても横8ドット，縦8ドットの大きさが必要になります。そのためテキスト・モードでひらがなを表示させると右側の2ドットが表示できません。MSX-BASICではこのことを避けるためにグラフィックⅠモードを通常のテキストの表示に使用しています。

①パターン・ジェネレータ・テーブル

テキスト・モードではパターン・ジェネレータ・テーブルに文字のパターンを入れます。ただしテーブルに設定したパターンの下位2ビットは表示されないので注意が必要です(**図2-24**)。設定できるパターンは256種類で，テーブルの大きさは8バイト×256＝2048バイトになります。パターン名称は，パターン・ジェネレータ・テーブルに定義した順番で決定され，テーブルの先頭から8バイト区切りで付けられた通し番号がその名称になります。

図2-24　パターン・ジェネレータ・テーブル

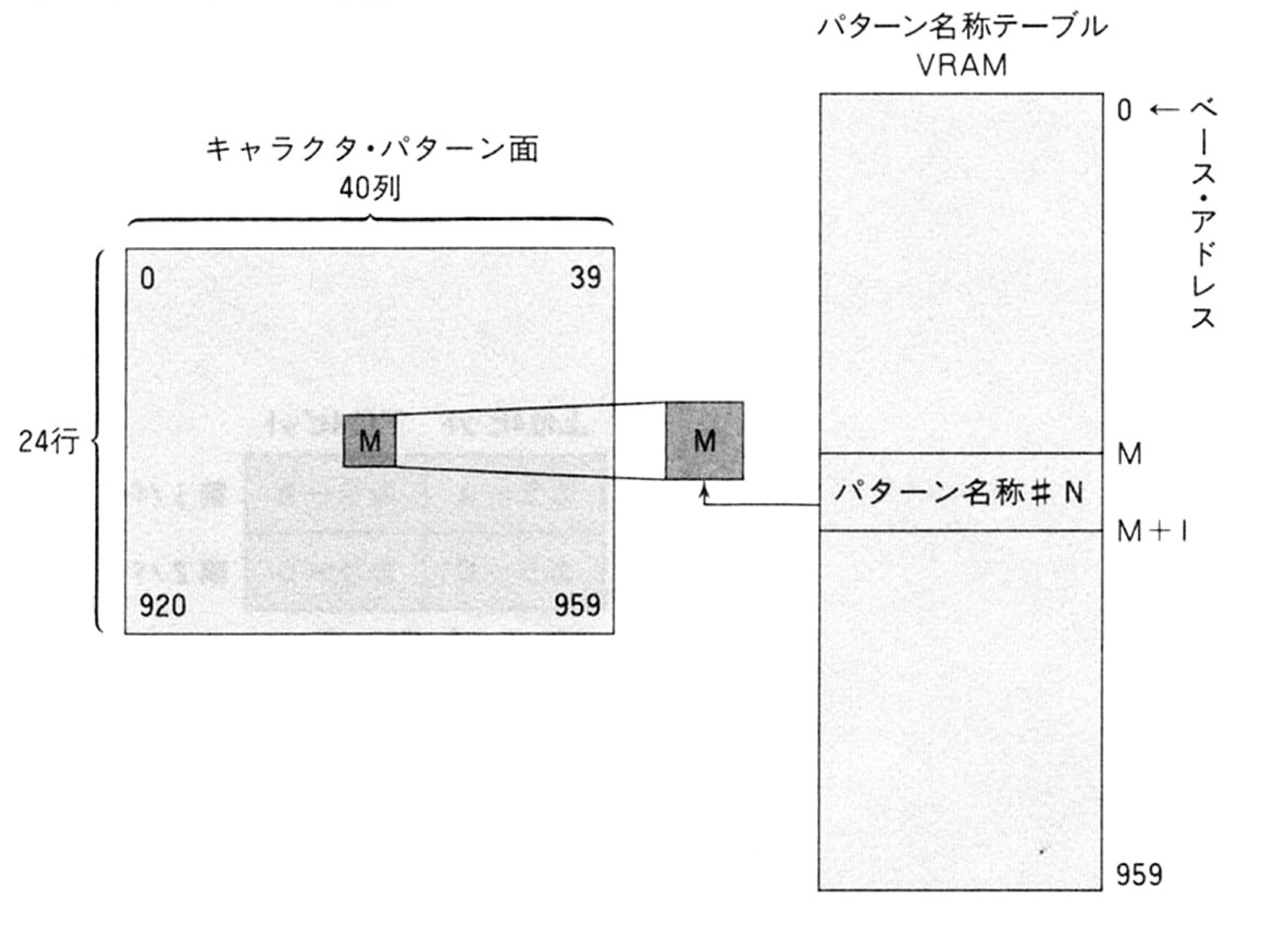


②パターン名称テーブル

テキスト・モードのパターン名称テーブルは**図2-25**のようになります。テーブルの大きさは40×24＝960 バイト必要です。このパターン

名称テーブルの使い方は，他のモードの場合と同じで，パターン名称テーブルの画面の位置と対応した番地に表示させたいパターンの名称を書き込めばよいわけです。

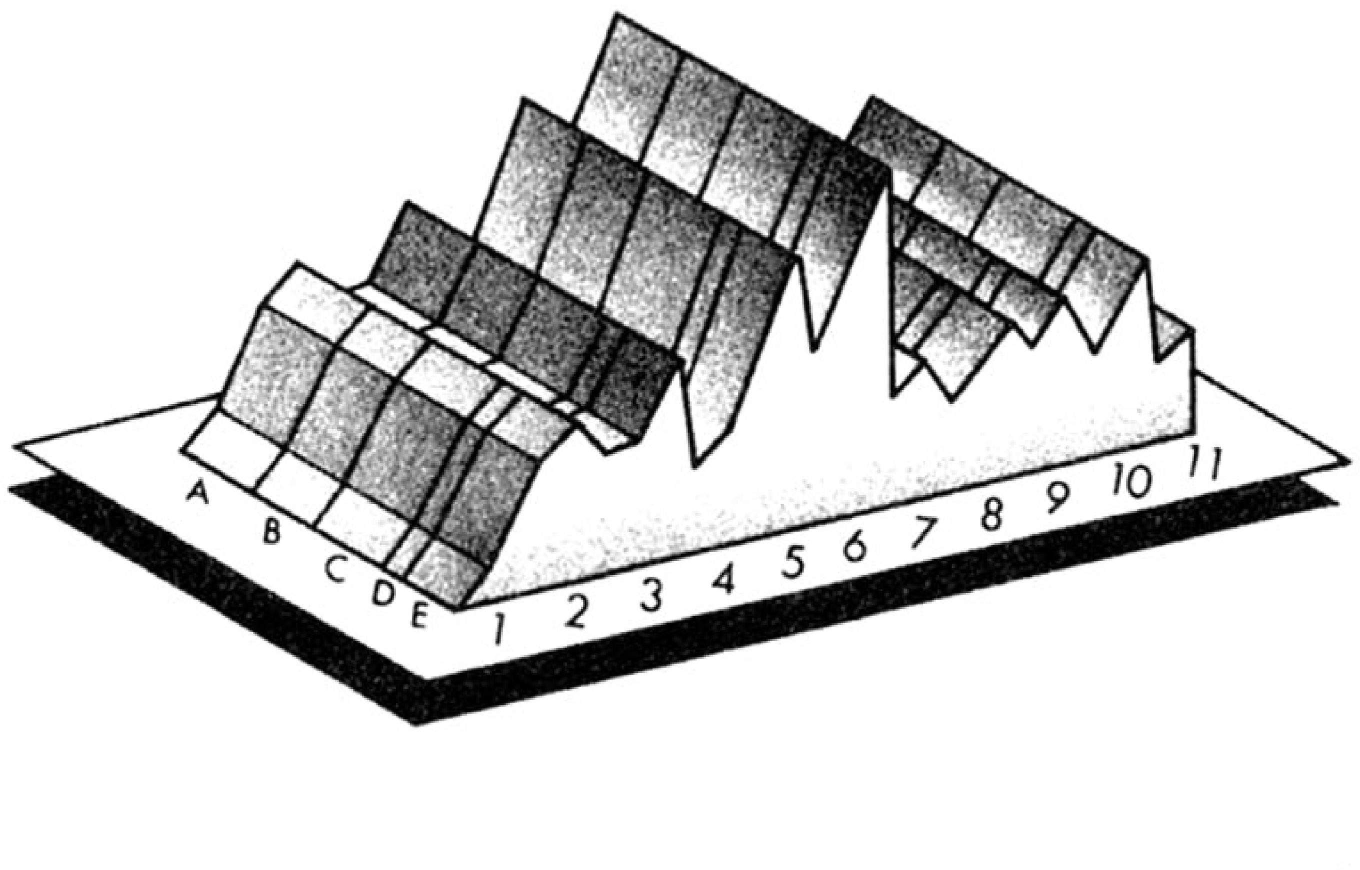
図2-25　パターン名称テーブル



③テキスト・モードのカラー指定

テキスト・モードのカラー指定は，ＶＤＰのレジスタ＃７で行います。レジスタ＃７の上位４ビットがパターンの中で“１”の部分の色指定，下位４ビットが“０”の部分の色指定です。また下位４ビットは，同時にバック・ドロップ面の色を指定します。

このことからわかるように，テキスト・モードでは１画面中２色しか使うことができません。

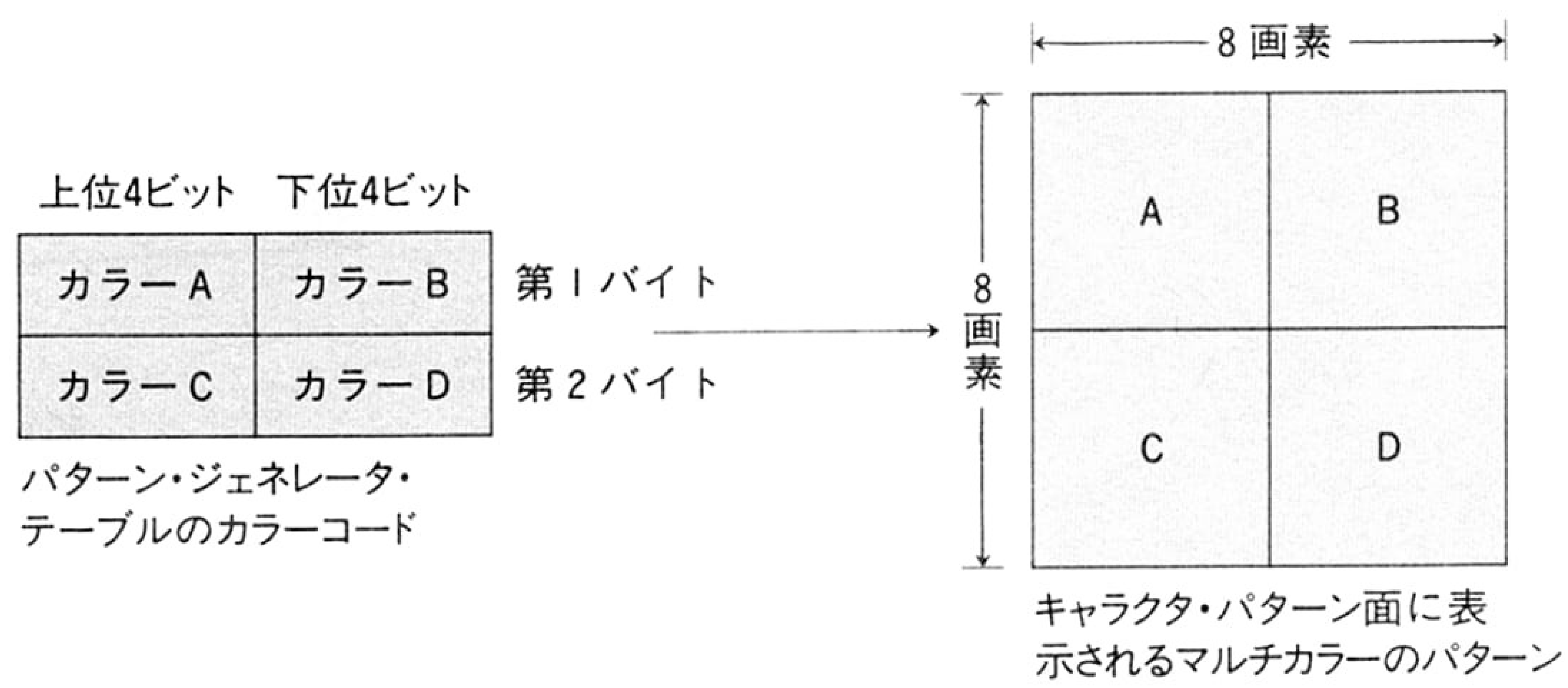


4. マルチカラーモード

マルチカラーモードでは**図2-27**のようなマルチカラーパターンを32列×24行に表示します。マルチカラーパターンは4つのブロックで構成され，それぞれのブロックごとに16色を指定することが可能です。

グラフィックIIモードに比べて，グラフィックの解像度が低い（64×48ブロック）かわりに，1ブロックごとの色を指定できることが特徴でスプライトも使えます。

図2-27　マルチカラー・パターン



①パターン・ジェネレータ・テーブル

パターン・ジェネレータ・テーブルは8バイト単位で構成され，テーブルの先頭に近い方から順にパターン名称が与えられます(**図2-28**)。マルチカラーモードでは，このパターン・ジェネレータ・テーブルに先程のマルチカラーパターンを定義しますが，これは1つのパターン名称につき4組用意します。このマルチカラーパターンは，そのパターン名称を表示する行によって4組のうちの1つが選択され，表示されるようになっています。

図2-29を例にとって説明しましょう。4組のマルチカラーパターンのうちどれが表示されるかは，表示する行を4で割った余りによって決定されます。したがってパターン名称0を8行目に表示した場合には，行を4で割った余りが0ですから，先頭のマルチカラーパターンがその位置に表示されます。9行目に表示した場合には，2つ目のマルチカラーパターンが表示されます。

パターン名称は0から255まであるので，テーブルの大きさは 8×256＝2048バイト必要です。

図2-28　パターン・ジェネレータ・テーブル

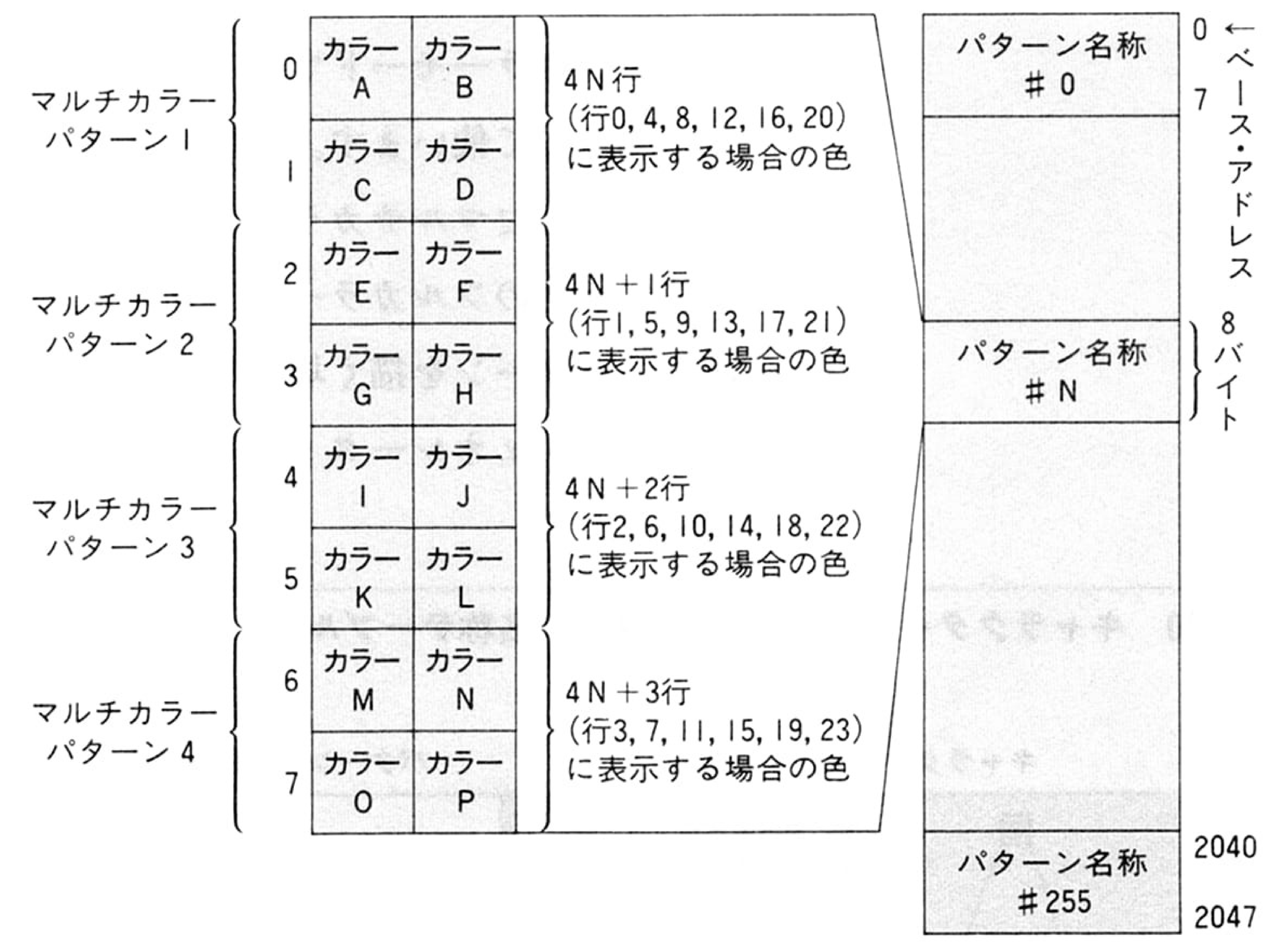
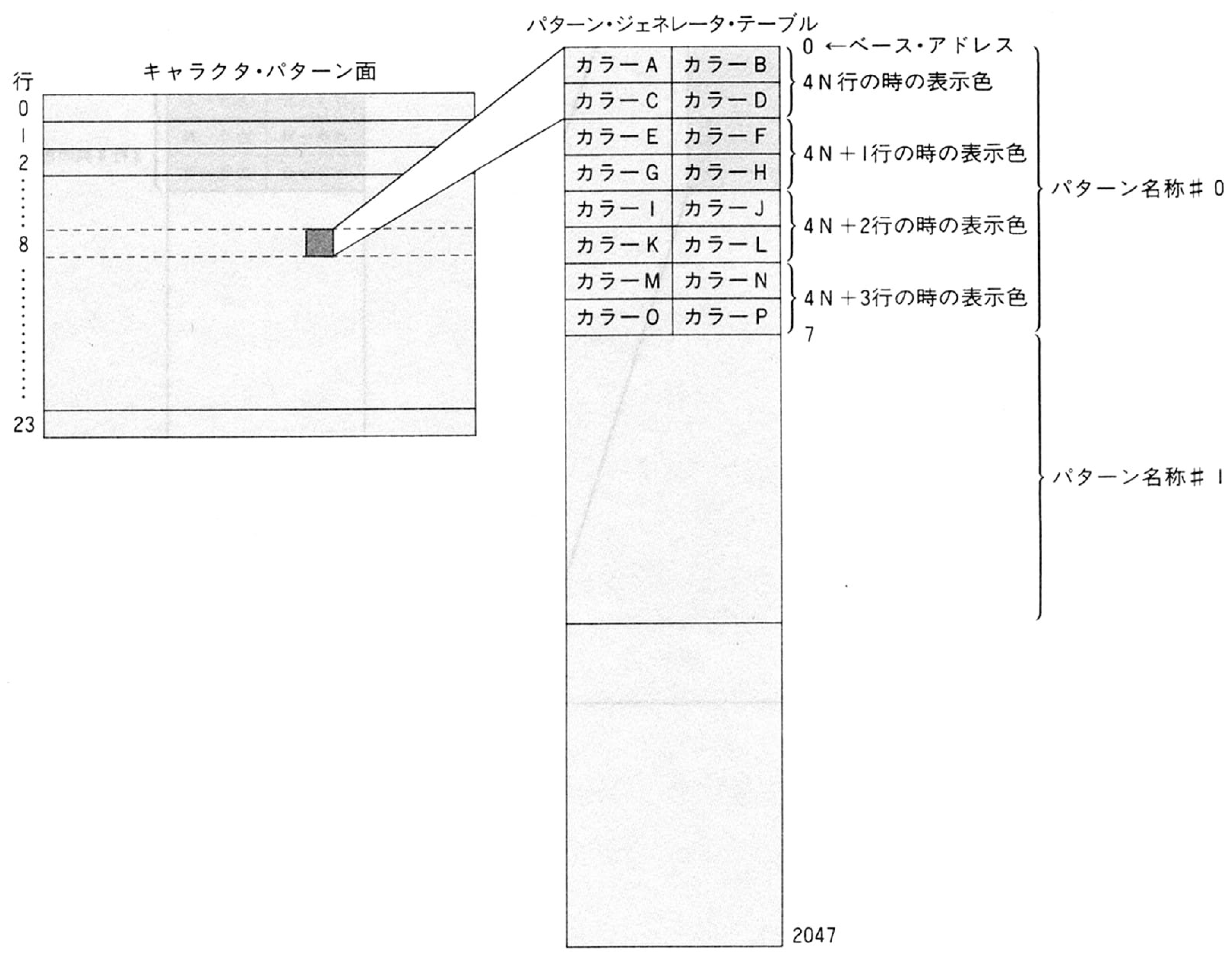


図2-29　表示位置と表示されるマルチカラーパターン

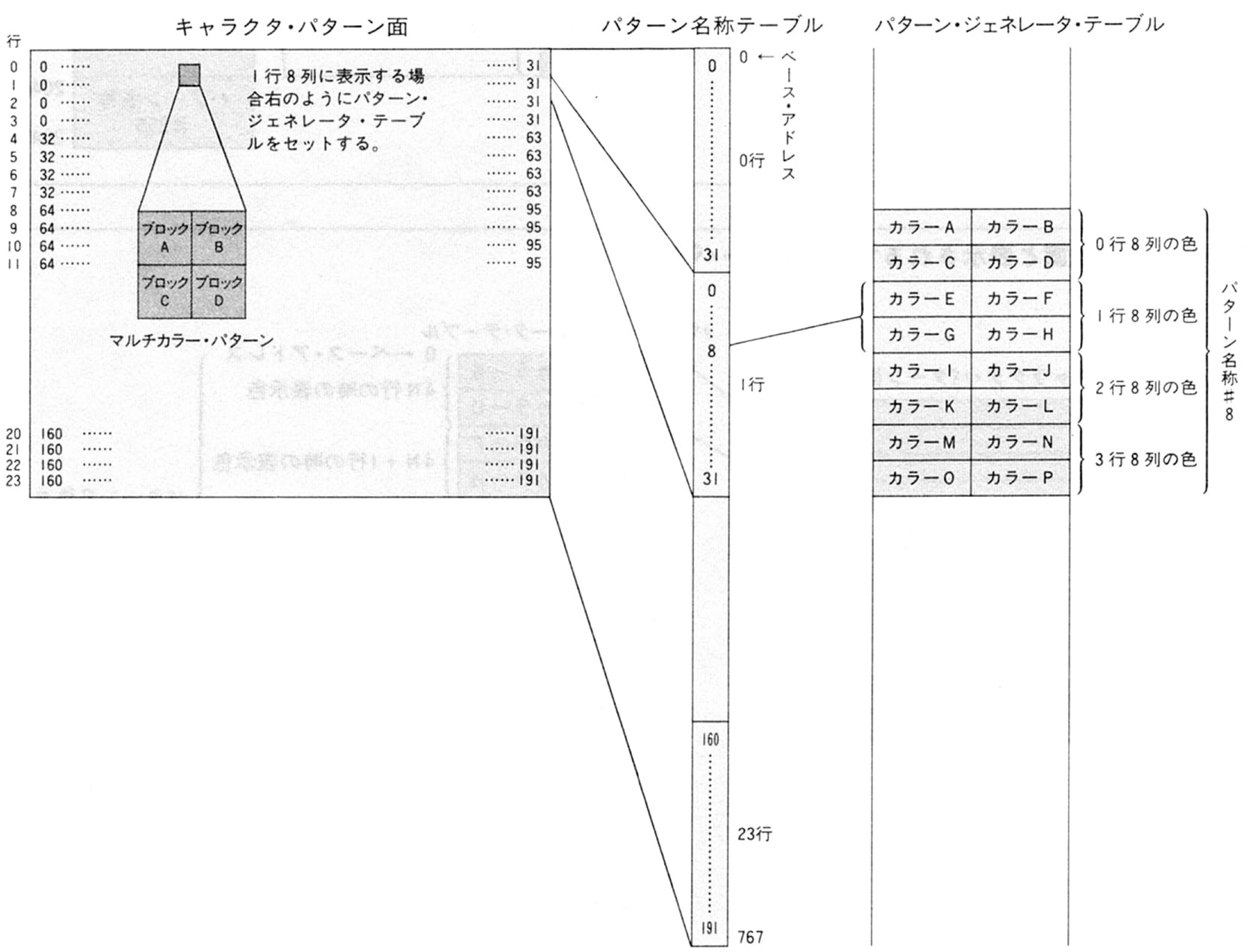


②パターン名称テーブル

パターン名称テーブルは，他のモードの場合と同じく32列×24行で構成され，大きさは768バイトです。

マルチカラーモードでは，このパターン名称テーブルを**図2-30**のように固定して使います。こうするとパターン・ジェネレータ・テーブルに定義したマルチカラーパターンと画面とが1対1に対応し，68×48ブロックのフルカラーグラフィックを実現することができます。画面上にパターンを描く場合には，パターン名称テーブルではなく，パターン・ジェネレータ・テーブルに定義したマルチカラーパターンを変更します。

図2-30　キャラクタ・パターン面とパターン名称テーブル



MSXのグラフィック機能を使いこなす

5 スプライト

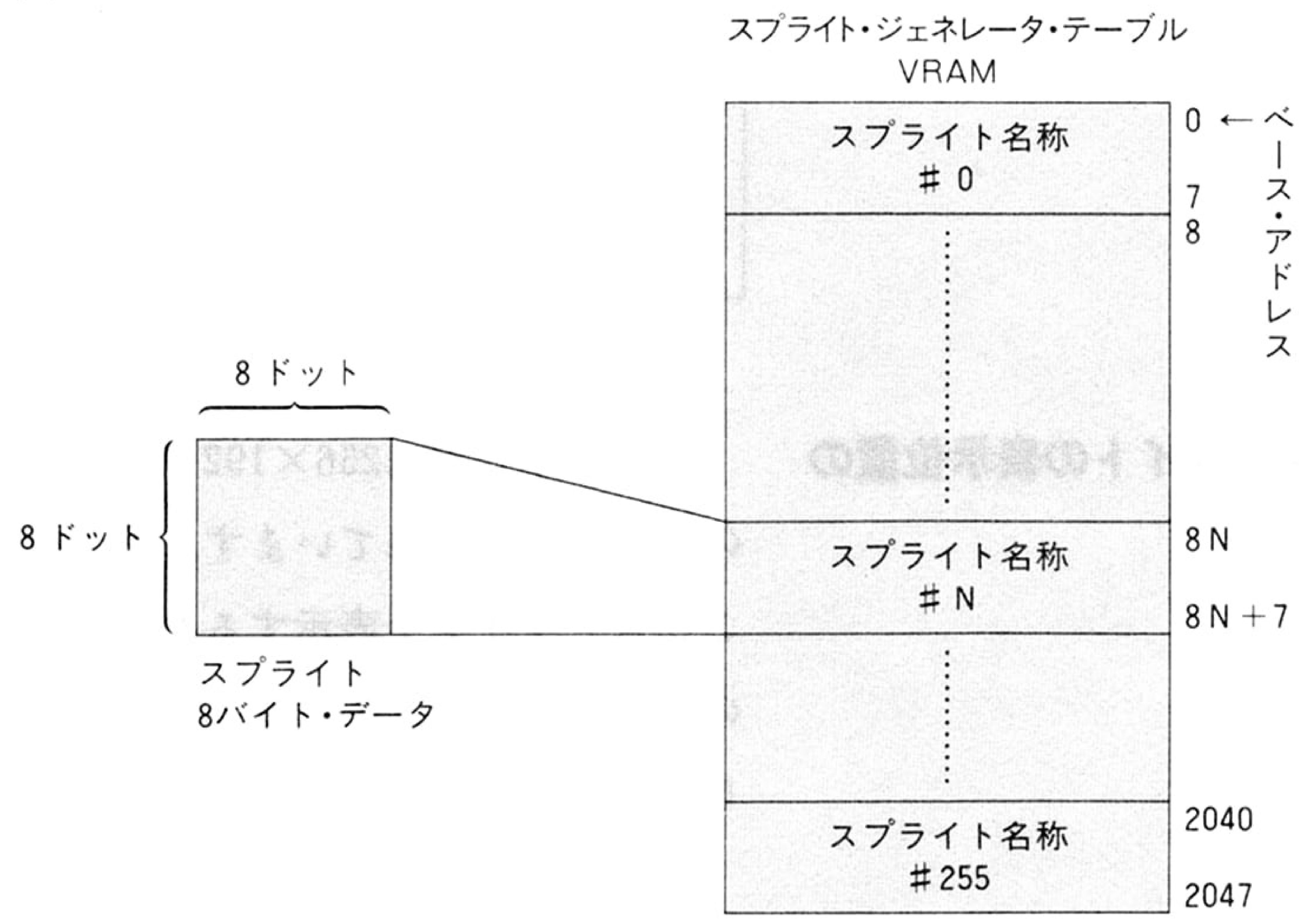
VDPの最も特徴的な機能といえるのがスプライトでしょう。このスプライトを使うためには，スプライト・ジェネレータ・テーブル，スプライト属性テーブルなどについて知らなければなりません。以下ではこれらを順に説明していきます。

1. スプライト・ジェネレータ・テーブル

スプライト・ジェネレータ・テーブルは，表示したスプライトのパターンを格納する領域です。ここには８×８のパターンなら256種類までを定義することができ，2048バイトの大きさを持ちます。

テーブルへのパターンの定義の仕方は，パターン・ジェネレータ・テーブルと同じで**図2-31**のようになります。スプライト・ジェネレータ・テーブルへパターンを定義すると，格納したアドレスにより自動的にスプライト名称が決まります。以後このスプライト名称がそのパターンを代表します。

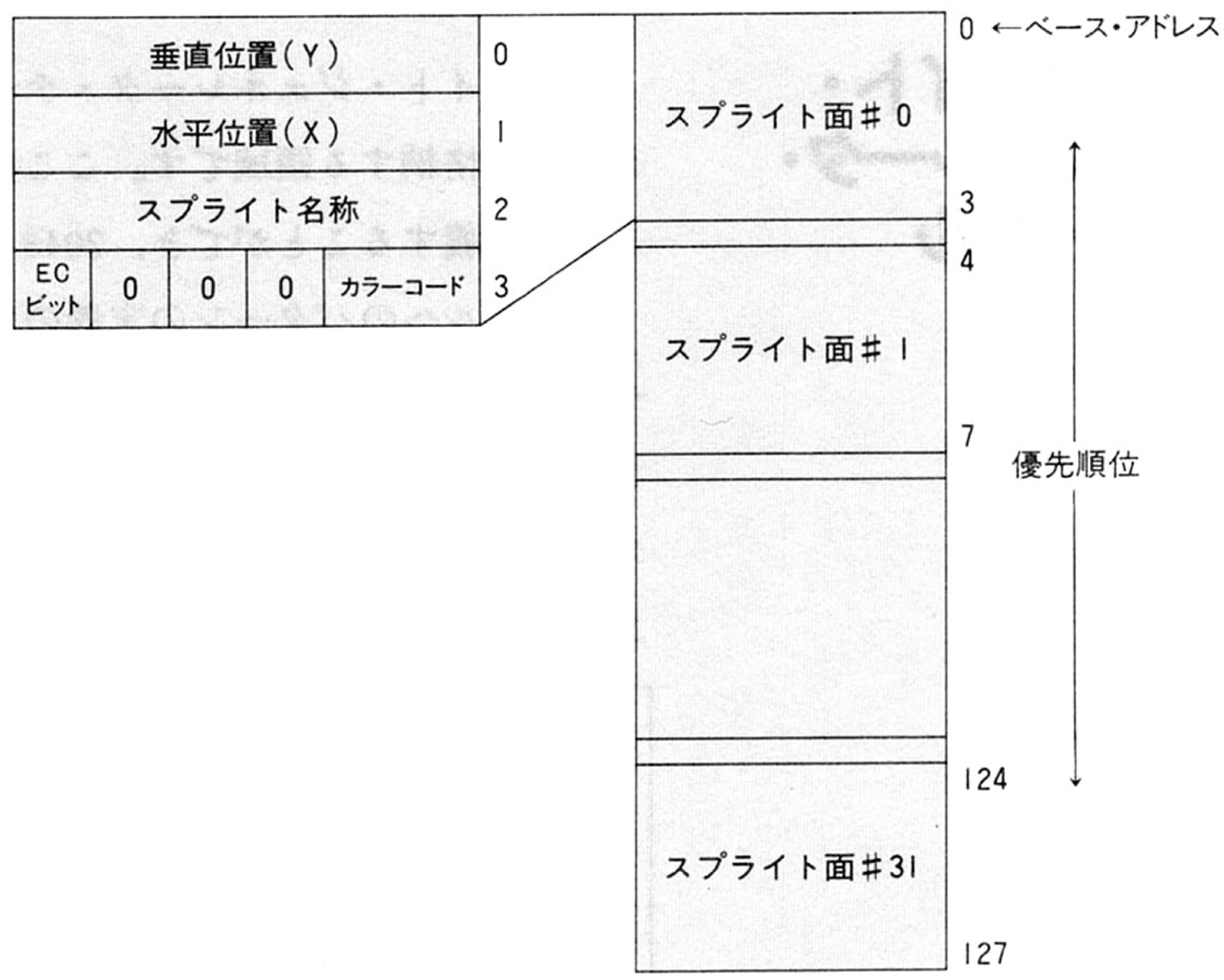
図2-31　スプライト・ジェネレータ・テーブル



2. スプライト属性テーブル

スプライト属性テーブルは，表示するスプライトの座標，スプライト名称，カラーなどを指定する領域です。スプライト属性は**図2-32**のように4バイト一組で構成され，それぞれが32枚のスプライト面と対応を持ちます。このスプライト面には表示のための優先順位があり，＃0が最高の優先順位を持ち，以下＃1，＃2と続きます。したがって最初の方の属性テーブルで示されるスプライトほど優先順位が高くなります。

図2-32　スプライト属性テーブル



①スプライトの表示位置の指定

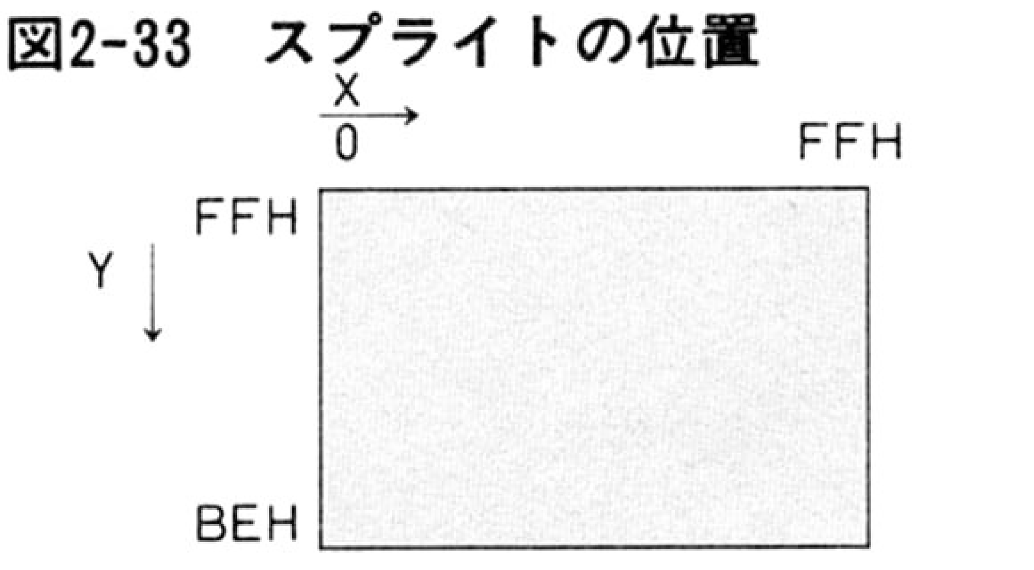
スプライト面は256×192ドットの大きさで構成され，座標は**図2-33**のように与えられています。垂直位置，水平位置には，この座標のどこにスプライトを表示するかをセットします。ここでいうスプライトの表示位置とは，スプライト・パターンの左隅が表示される点を意味します。

さて図を見てわかるように，画面の最上端の垂直位置が－1(FFH)で，最下端が190(BEH)になっています。これは一般の感覚と違うので，スプライトの垂直位置を指定する場合には注意が必要です。また表示するスプライトの一部または全部がこの領域をはみだした場合は，はみ出た部分についての画面表示は行われません。

一方，水平位置については画面の左端が0で，右端が255(FFH)になります。表示するスプライトが画面の右端からはみ出るとはみ出し

た部分は画面の左端に表示されます。さらに，垂直位置に208(D0H)を書き込むと，それより優先順位の低いスプライト面は表示されなくなります。

図2-33　スプライトの位置



②スプライト名称の指定

スプライト・ジェネレータ・テーブルで定義したスプライトの名称番号をスプライト名称に入れます。

③スプライトのカラー指定

スプライト・パターンのビットが“1”の部分の色をスプライト属性テーブルのカラーコードにセットします。1つのスプライトにつき指定できる色はこの1色だけで，パターンが“0”の部分は常に透明です。

④ECビット

ECビットというのはEarly Clock ビットの略で，このビットを1にするとスプライト面が左に32ビット分シフトします。これはそのスプライトを画面に表示したくないという場合，スプライトを画面から追い出すために使用します。

3. スプライト・サイズ

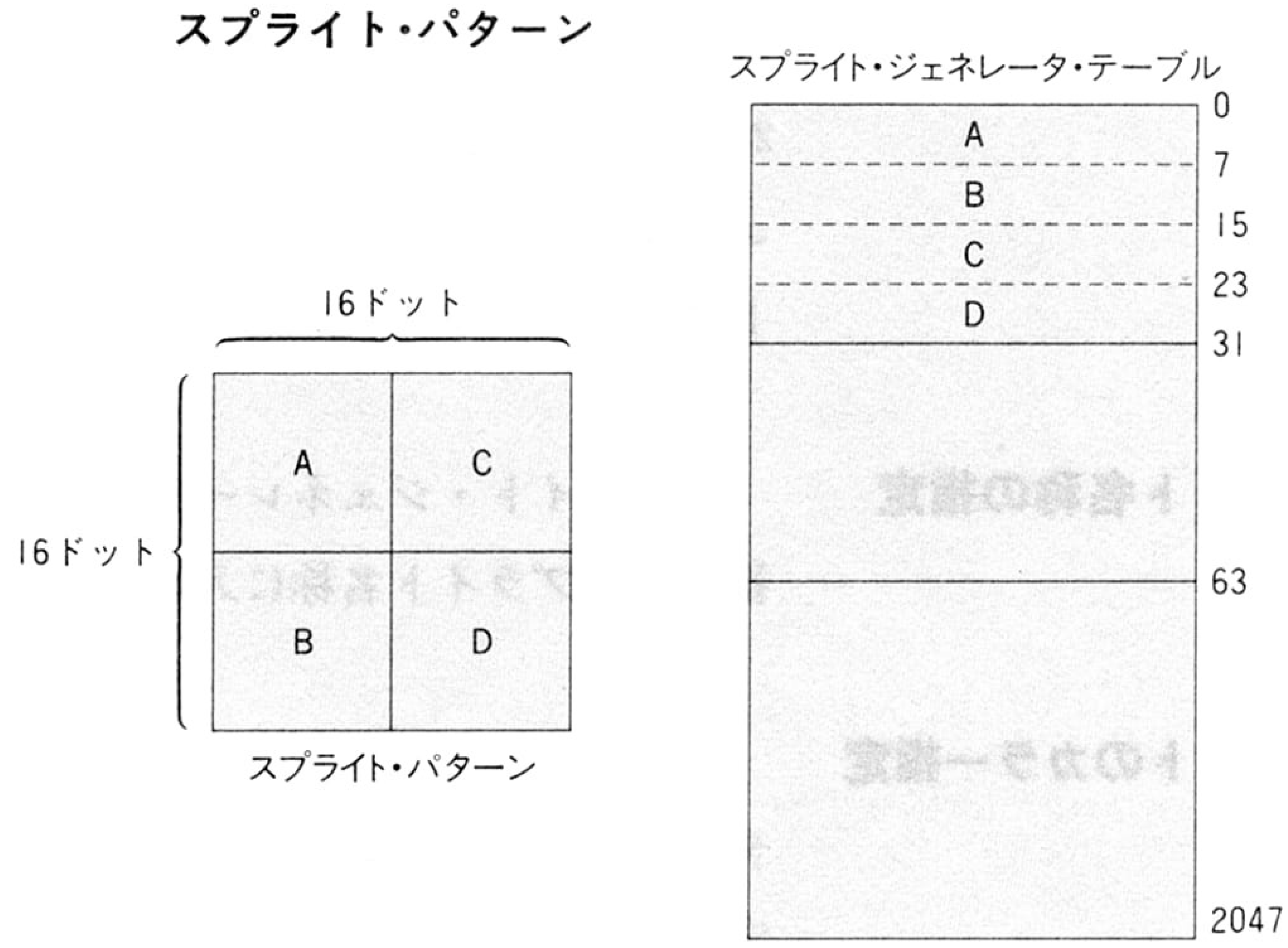
スプライトのサイズには，8×8ドットまたは16×16ドットの大きさのどちらか1つを選ぶことができます。サイズの指定はレジスタ#1のSIZEビットで行い，ここに“1”を書き込むとスプライト・サイズは16×16になります。両方のサイズのスプライトを混在して使用することはできません。ここではいままで触れていなかった16×16のスプライトの取り扱いについて解説することにしましょう。

16×16ドットのスプライトでは，1つのスプライトあたり32バイトの大きさのパターンが必要になります。このため，スプライト・ジェネレータ・テーブルでは，8×8ドットの場合のパターンを4つ分つなげて使います(**図2-34**)。

スプライト名称は**図2-34**の例からいうと，Aが0，Bが1，Cが2，Dが3になるのですが，16×16のスプライトの場合は，スプライト属性テーブルの名称にこの4つのうちのどれか1つを入れます。

それから当然のことながら，16×16ドットの大きさのスプライトを使う場合は，定義できるスプライトの数が8×8の場合の256種類から，64種類に減ってしまいます。

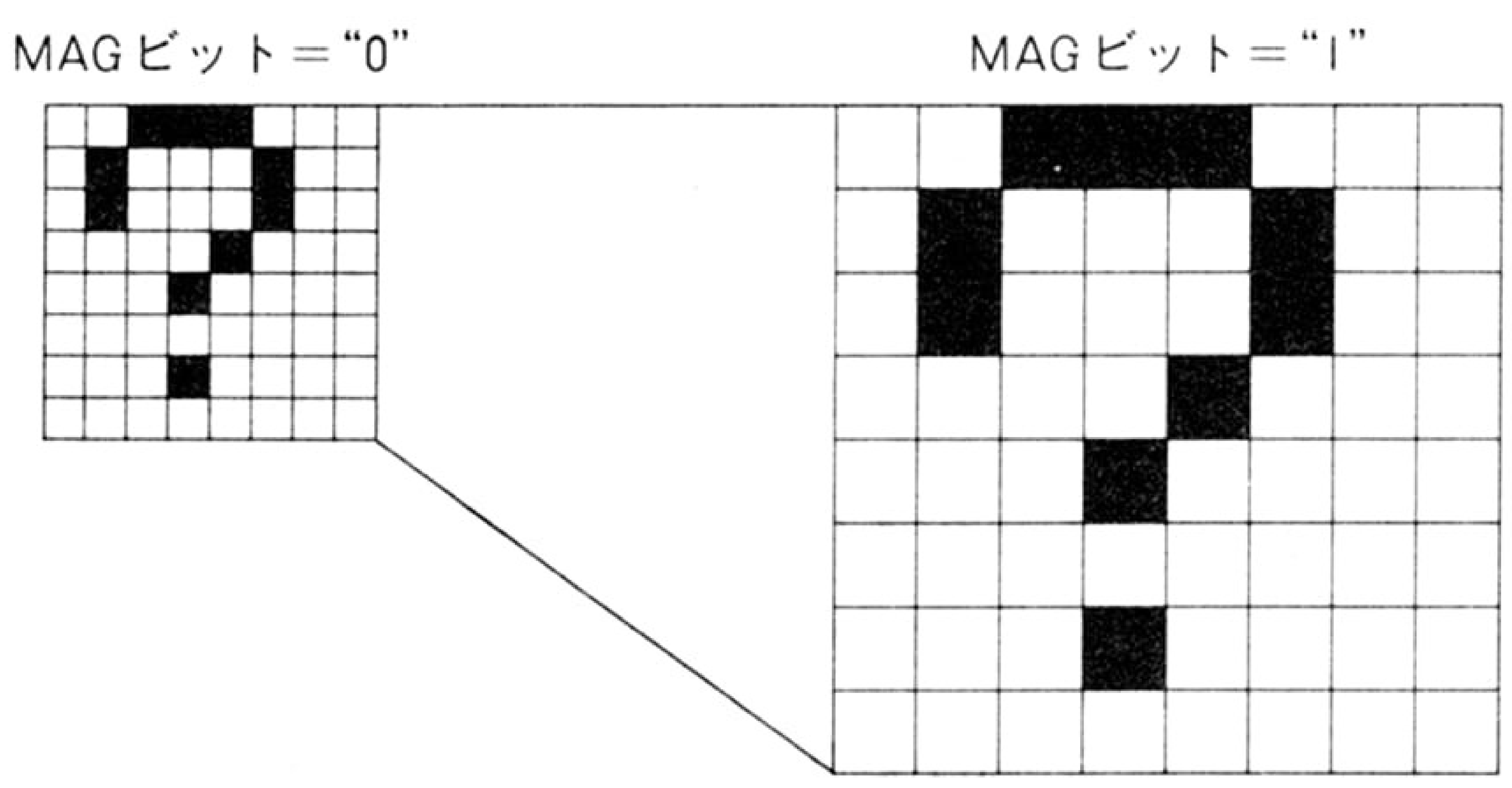
図2-34　スプライト・ジェネレータ・テーブルと
スプライト・パターン



4. スプライトの拡大表示

レジスタ＃1のMAGビットに1を書き込むと，スプライトの大きさが縦横両方向に2倍に拡大されます。ただし，これはスプライトの解像度が上がるということではなく，単にスプライトを2倍の大きさに引き伸ばして表示するということを意味します(**図2-35**)。

図2-35　スプライトの拡大表示



5. スプライトの衝突

スプライトどうしの衝突が起こったかどうかをステータス・レジスタの衝突フラグ(C)で調べることができます。衝突したかどうかは，パターンの1の部分が衝突したかどうかで決まり，衝突していればフラグがセットされます。パターンが0の部分について衝突が起こってもフラグはセットされません。

このフラグはゲームを作る場合に有効で，ゲームのキャラクタをすべてスプライトに定義した場合は，いちいちキャラクタの座標を調べ

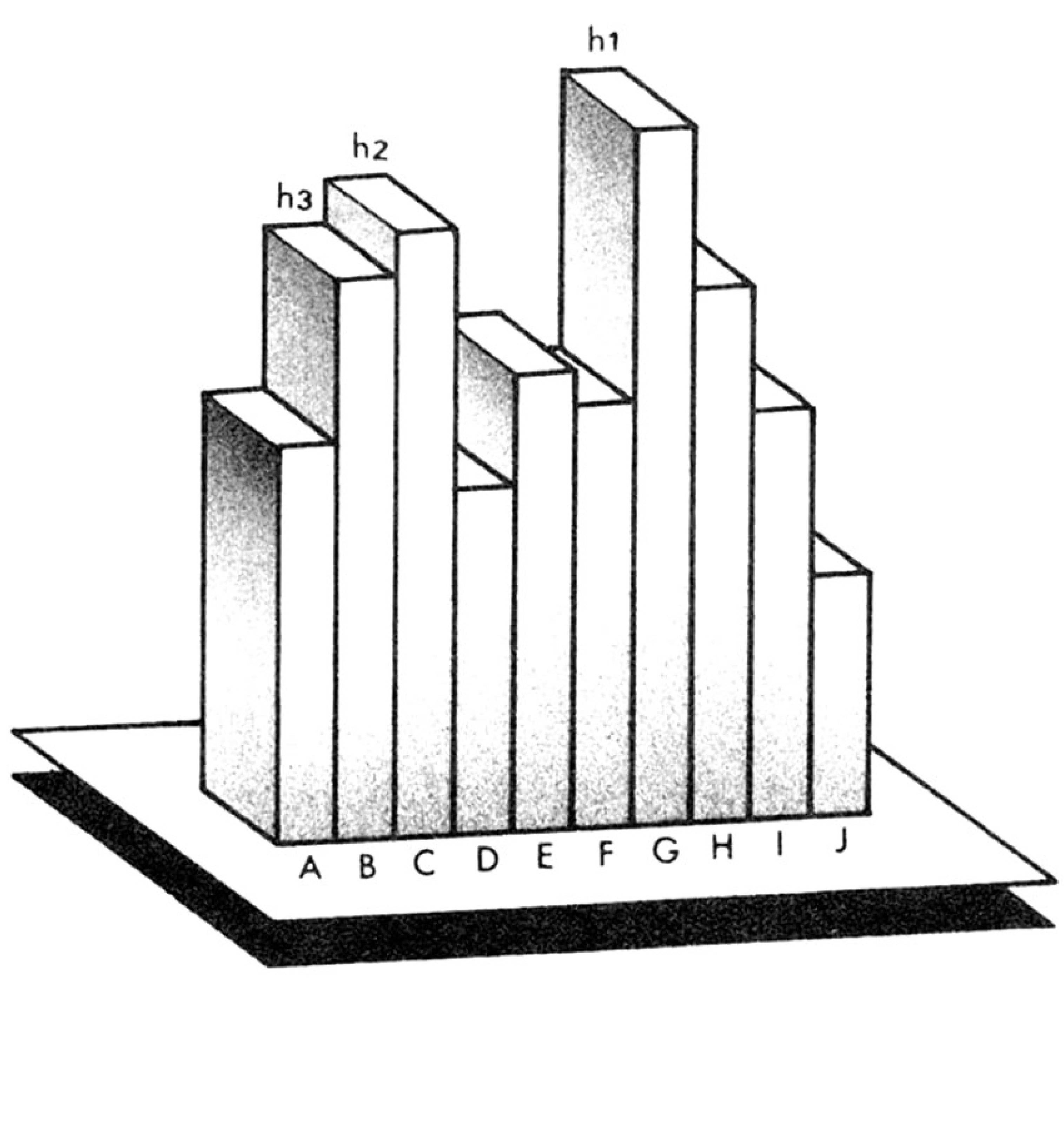
なくても衝突の有無を知ることができます。

6. スプライトの表示制限

スプライトは非常に便利な機能ですが，表示に関して次のような制限があり，使用にあたっては注意が必要です。

スプライトは同一水平ライン上に4個までしか存在できません。5個以上のスプライトが存在すると優先度の高い4個のスプライトが表示され，5番目以降のスプライトは透明になって表示されません。したがってプログラムの側では，同一水平ライン上に5個以上のスプライトが並ばないような工夫をするか，仮りにそうした事態が発生しても，すぐに対処できるようにしておく必要があります。

このようなスプライトの状態は，ステータス・レジスタによって知ることができます。もし5個以上のスプライトが同一水平ライン上に並ぶと，5Sビットが1にセットされ，同時に表示されなかった5番目のスプライト面の番号がステータス・レジスタの下位5ビットに現われます。



MSXのグラフィック機能を使いこなす

6 VDPへのアクセス

1. ハードウェアによるVDPとの接続

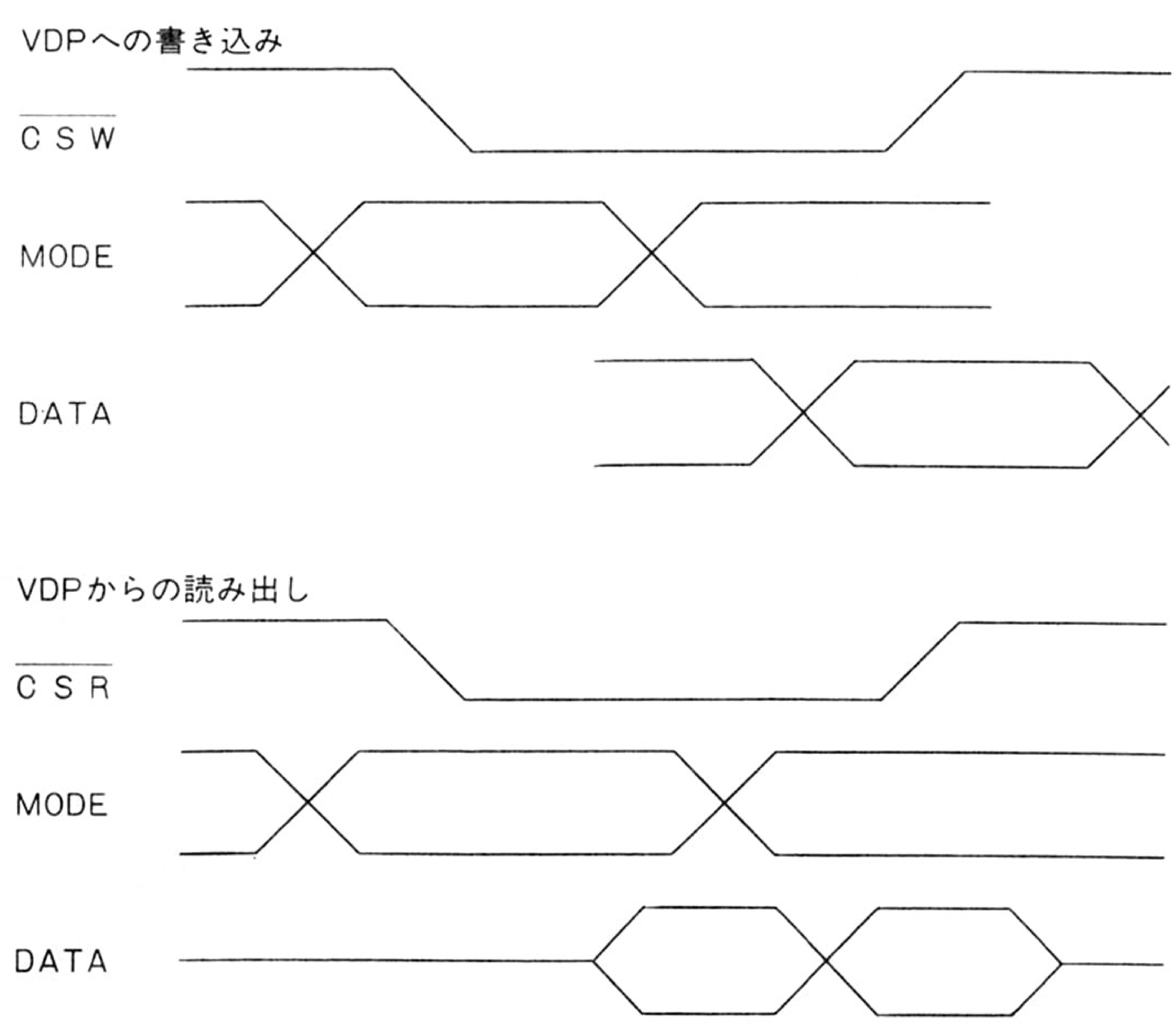
ＶＤＰにはＭＯＤＥ，$\overline{\text{ＣＳＷ}}$，$\overline{\text{ＣＳＲ}}$という３本の信号線があり，これ３本の線でＶＤＰのコントロールを行います。ＭＯＤＥは，書き込みあるいは読み出しがＶＤＰのレジスタに対するものかＶＲＡＭに対するものかを指定するもので，これがＬレベルのときＶＲＡＭが指定されます。$\overline{\text{ＣＳＷ}}$と$\overline{\text{ＣＳＲ}}$は，ＶＤＰに対するチップ・セレクトであると同時に読み・書きを指定します。

ＭＳＸでは，アドレス・バスのA_{11}～A_{15}と$\overline{\text{ＩＯＲＱ}}$により，チップセレクト信号を作り出します。この信号と$\overline{\text{ＲＤ}}$のＯＲを取ったものを$\overline{\text{ＣＳＲ}}$に，$\overline{\text{ＷＲ}}$とＯＲを取ったものを$\overline{\text{ＣＳＷ}}$に継いでいます。

一方，ＭＯＤＥについてはA_0をそのまま接続します。したがって，I/Oポート98Ｈへの読み書きは，ＶＲＡＭへのデータの読み書きを意味し，99Ｈへの読み書きはＶＤＰのレジスタの読み書き，ＶＲＡＭのアドレスのセットを意味します。

これらＶＤＰのアクセス時のタイミング・チャートは**図2-36**のようになります。

図2-36 CPU-VDP タイミング・チャート



2. レジスタへの書き込み

最初にレジスタへ書き込もうとするデータをI/Oポート99Hに出力し，続けてレジスタの番号(0～7)に80Hを加えた値を同じくI/Oポート99Hに出力します。CPUのAレジスタにデータ，Cレジスタにレジスタ番号が入っていたとすると，実際のプログラムは次のようになります。

```
OUT   (99H), A………データを送る
LD    A, C
ADD   A, 80H…………レジスタ番号に80Hを加える
OUT   (99H), A………レジスタを指定する
```

ポート99Hの出力は，必ず連続して行われなくてはなりません。したがって，割り込みの発生する可能性があるときは，ポートへの出力に先立って割り込みを禁止しておく必要があります。

3. レジスタからの読み出し

VDPのレジスタ#0～#7は書き込み専用レジスタなので，内容を読み取ることはできません。したがってCPUによる読み出し対象となるのはステータス・レジスタということになります。ステータス・レジスタはI/Oポート99Hに対する入力命令で，その内容を読み取ることができます。

```
IN A, (99H)……Aレジスタにステータス・レジスタの内容
              が入る
```

4. VRAMへの書き込み

まず，VDPに対して書き込みを行うVRAMのアドレス(0000H～3FFFH)を指定します。アドレスの指定は，I/Oポート99Hに対してVRAMアドレスの下位8ビットを出力し，続けてアドレスの上位6ビットに40Hを加えた値を同じ99Hに出力して行います。

アドレスの指定が終わったら，つぎにデータを送りますが，これは単にI/Oポート98Hにデータを出力するだけです。ただし，VRAMのアドレスを指定する際に，VDPが数μsの時間を必要とするので，データの出力をする前にウェイトを入れてやる必要があります。

DEレジスタにVRAMのアドレス，Cレジスタにデータが入っていたとすると，プログラムは次のようになります。

```
LD    A, E
OUT   (99H), A  ……指定するアドレスの下位8ビットを出力
LD    A, D
ADD   A, 40H  ………アドレスの上位6ビットに40Hを加える
OUT   (99H), A  ……アドレスの上位6ビットを指定する
EX    (SP), HL ……VDPにアドレスがセット
EX    (SP), HL ……されるまでのウェイト
LD    A, C
OUT   (98H), A  ……データを送る
```

VRAMに対して，連続してデータの書き込みを行いたい場合は，98Hへの出力を繰り返せばアドレスは自動的にインクリメントされていきます。

またこうした一連の動作は，VDPのレジスタに対する書き込みの場合と同様に必ず連続して行われなければなりません。

5. VRAMからの読み出し

VRAMに対する書き込みの場合と同様，まず書き込みを行うVRAMのアドレスを指定します。最初にアドレスの下位8ビットをI/Oポート99Hに出力したあと，続けて上位6ビットをそのまま99Hに出力します。

こうしてアドレスの指定をしたあとで，ポート98Hに対して入力命令を実行すればVRAMの内容を読み取ることができます。次の例はDEレジスタにVRAMのアドレスが入っていた場合のプログラムです。

```
LD    A, E
OUT   (99H), A  ……指定するアドレスの下位8ビットを出力
LD    A, D
OUT   (99H), A  ……残りの6ビットを指定する
EX    (SP), HL ……VDPにアドレスがセット
EX    (SP), HL ……されるまでのウェイト
IN    A, (98H)  ……データの読み込み
```

ＶＲＡＭから連続してデータを読み取りたい場合は，ポート98Hに対する入力命令を繰り返します。その際，アドレスは自動的にインクリメントされていきます。

またＶＲＡＭからの読み込みの場合についても，ＶＤＰに対する一連の入出力動作は，必ず連続して行われなければなりません。したがって必要に応じて割り込みのマスク等の処置をしておかなくてはなりません。

MSXのグラフィック機能を使いこなす

7 VDPに関するBIOSルーチン

これまで説明してきたようなＶＤＰへのアクセス方法は，実際のプログラムを組むにあたって，ユーザーが直接行うものではありません。なぜなら，こうしたＶＤＰのレジスタに関する書き込みなどは，全てBIOS中に用意されたルーチンによってサポートされているからです。

したがって特別な理由でもない限り，今後ＶＤＰへのアクセスは全てこのBIOSを経由して行います。この特別な理由に相当するものとしては，ＶＤＰによる高速処理が要求されるときなどで，その場合はメモリの6番地および7番地に格納してあるＶＤＰのポート番号（通常98Hが入っている）を調べて，そのポートにアクセスを行います。

ＶＤＰをサポートするＢＩＯＳルーチンは**表2-4**のとおりで，ここではこのうちの主なルーチンについて，使用法を簡単に解説しておきましょう。

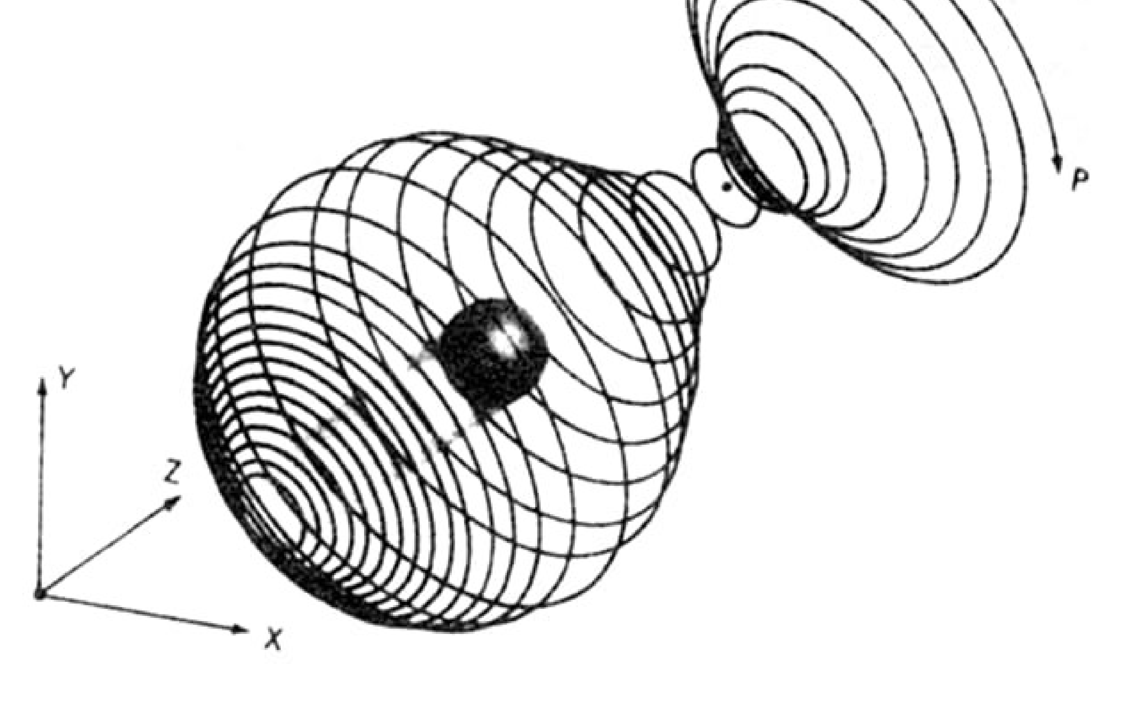


表2-4　VDPに関するBIOSルーチン

番　地	ルーチン名	内　容
0041	DISSCR	画面表示をオフにする
0044	ENASCR	画面表示をオンにする
0047	WRTVDP	VDPのレジスタにデータを書き込む
004A	RDVRM	VRAMからデータを読む
004D	WRTVRM	VRAMにデータを書く
0050	SETRD	VDPを読み込み状態にする
0053	SETWRT	VDPを書き込み状態にする
0056	FILVRM	VRAMを一定の値でうめる
0059	LDIRMV	VRAMの内容をメモリへブロック転送
005C	LDIRVM	メモリの内容をVRAMへブロック転送
005F	CHGMOD	VDPのモードをセットする
0062	CHGCLR	フォア・グランド，バック・グランド，ボーダーカラーを指定する
0066	NMI	NMI（ノンマスカブル・インタラプト）の処理を行う
0069	CLRSPR	スプライトを初期化する
006C	INITXT	画面をテキストモード(40×24)に初期化する
006F	INIT32	画面をテキストモード(32×24グラフィックⅠモード)に初期化する
0072	INIGRP	画面を高解像度グラフィック・モードに初期化する
0075	INIMLT	画面をマルチカラーモードに初期化する
0078	SETTXT	VDPのみをテキスト・モードにセットする
007B	SETT32	VDPのみをグラフィックⅠモードにセットする
007E	SETGRP	VDPのみをグラフィックスⅡモードにセットする
0081	SETMLT	VDPのみをマルチカラーモードにセットする
0084	CALPAT	スプライト・ジェネレータ・テーブルの番地を調べる
0087	CALATR	スプライト属性テーブルの番地を調べる
008A	GSPSIZ	スプライトのサイズを調べる
008D	GRPPTR	高解像度グラフィック画面に文字を出力する

1. WRTVDP (0047H)

このルーチンは，VDPのレジスタにデータを書き込むものです。CPUのCレジスタにVDPのレジスタ番号を入れ，Bレジスタに書き込む値を入れて，このルーチンをコールします。このルーチンをコールした後では，CPUのAFレジスタ，BCレジスタの内容は保証されません。

2. RDVRM (004AH)

VRAMの指定した番地の内容を読み込みます。HLに読み取ろうとするVRAMの番地を入れ，このルーチンをコールするとAレジスタに読み込んだ値が入ります。当然ながら，このルーチンをコールした場合はAFレジスタの内容が変更されています。

3. WRTVRM (004DH)

ＶＲＡＭの指定した番地にデータを書き込みます。ＨＬレジスタでＶＲＡＭの番地を指定し，Ａレジスタに書き込むデータを入れてこのルーチンをコールします。このルーチンをコールした場合，ＡＦレジスタの内容は保証されません。

4. LDIRMV (0059H)

ＶＲＡＭの内容をメイン・メモリ上へブロック転送します。パラメータは次の通りです。

> ＨＬ←ＶＲＡＭ上のデータの先頭番地
> ＤＥ←転送先のメモリの先頭番地
> ＢＣ←転送するデータの長さ

これらの値をＣＰＵのレジスタにセットした後，このルーチンをコールします。このルーチンをコールした後は，ＣＰＵの全てのレジスタの内容は保証されません。

5. LDIRVM (005CH)

メイン・メモリ上のデータをＶＲＡＭへブロック転送します。次のようにパラメータを指定します。

> ＨＬ←メモリ上のデータの先頭番地
> ＤＥ←転送先のＶＲＡＭの先頭番地
> ＢＣ←転送するデータの長さ

LDIRMVと同様に，全てのレジスタの内容は保証されません。

6. CHGMOD (005FH)

ＶＤＰのモードをセットします。モードはＳＣＲＭＯＤ(メイン・メモリのＦＣＡＦＨ番地の内容）をあらかじめ次のようにセットしておくことによって指定します。

> SCRMOD(FCAFH)←0：テキスト・モード
> 1：グラフィックⅠモード
> 2：グラフィックⅡモード
> 3：マルチカラーモード

全てのレジスタの内容は保証されません。

7. CHGCLR (0062H)

画面の色を指定します。パラメータは以下の通りです。

FORCLR(F3E9H)←フォアグラウンド・カラー
BACCLR(F3EAH)←バック・グラウンド・カラー
BDRCLR(F3EBH)←ボーダーカラー

全てのレジスタの内容は保証されません。

8. INITXT (006CH)

画面をテキスト・モード（40×24）に初期化します。次のようにして，各テーブルの先頭番地をセットしてからこのルーチンをコールします。

TXTNAM(F3B3H)←パターン名称テーブルの先頭番地
TXTCGP (F3B7H)←パターン・ジェネレータ・テーブルの先頭番地

全てのレジスタの内容は保証されません。

9. INI32 (006FH)

画面をグラフィックⅠモード（32×24のテキスト・モード）に初期化します。

T32NAM(F3BDH)←パターン名称テーブルの先頭番地
T32CGP(F3C1H)←パターン・ジェネレータ・テーブルの先頭番地
T32COL(F3BFH)←カラーテーブルの先頭番地
T32ATR(F3C3H)←スプライト属性テーブルの先頭番地
T32PAT(F3C5H)←スプライト・ジェネレータ・テーブルの先頭番地

全てのレジスタの内容は破壊されます。

10. INIGRP (0072H)

画面をグラフィックⅡモード（高解像度グラフィック・モード）に初期化します。

GRPNAM(F3C7H)←パターン名称テーブルの先頭番地
GRPCGP (F3CBH)←パターン・ジェネレータ・テーブル

の先頭番地
GRPCOL（F3C9H）←カラーテーブルの先頭番地
GRPATR（F3CDH）←スプライト属性テーブルの先頭番地
GRPPAT（F3CFH）←スプライト・ジェネレータ・テーブルの先頭番地

全てのレジスタの内容は破壊されます。

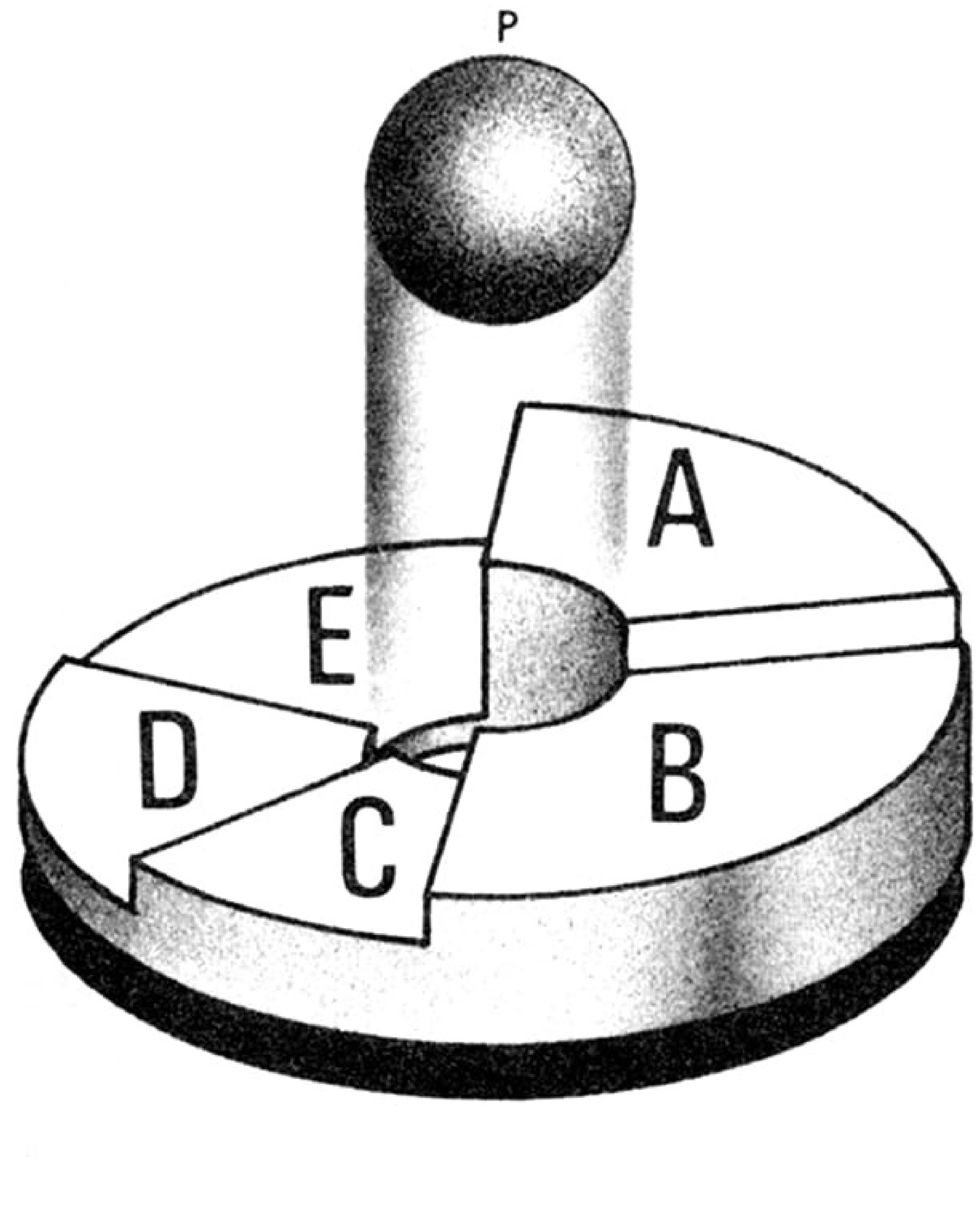
11. INIMLT（0075H）

画面をマルチカラーモードに初期化します。

MLTNAM（F3D1H）←パターン名称テーブルの先頭番地
MLTCGP（F3D5H）←パターン・ジェネレータ・テーブルの先頭番地
MLTCOL（F3D3H） カラーテーブルの先頭番地
MLTATR（F3D7H）←スプライト属性テーブルの先頭番地
MLTPAT（F3D9H）←スプライト・ジェネレータ・テーブルの先頭番地

全てのレジスタの内容は破壊されます。

以上がＶＤＰに関する主なＢＩＯＳルーチンです。



MSXのグラフィック機能を使いこなす

8 VDPの応用

ここでは，ＶＤＰを使った実際例とゲームへの応用例を取り上げます。ゲームで使う表示モードは，やはりグラフィックＩかIIモード（特にグラフィックの美しさを強調したい場合は，IIモードにならざるを得ないでしょう）ということになると思われますが，ここでも主にグラフィックIIモードを使っていきます。

1. パターン名称(ID)テーブルの考察

キャラクタ・パターン面にグラフィック・パターンを表示するためには，パターン・ジェネレータ・テーブルとカラーテーブルにそれぞれパターン・データとカラーデータを送る必要があります。さらに表示したい位置を名称テーブルに書き込むという作業をします。

リスト2-1 は，その１例ですが，1180行にわざとウェイト（時間待ち）を入れています。パターンとカラーのデータは書き込む位置(VRAMのアドレス)によって，自動的に名称が決まってしまいます。

リスト2-1 ではデータを書き込むとともに画面の左上端（ブロック１の名称０）にパターンが表示されてしまいます。これを防ぐには，次の２つの方法があります。

①パターンやカラーデータを書き込む前に，名称テーブルを書き変える。

②ＶＤＰは画面表示のON/OFFができるので，データを書き込む前に表示OFFにしておく。

②はＢＩＯＳでもサポートしているので，こちらの方が簡単に実現できます。**リスト2-1**に**リスト2-2**の1091，1092と，1191，1192の４行を加えてください。

リスト2-1

```
1000 '   PROGRAM    PATTERN ID TABLE TEST
1010 '   DATE       19-JUN-84
1020 '
1030 '   CONSTANTS
1040 PG=&H0       :'PATTERN GENERATOR TABLE
1050 CT=&H2000    :'COLOR TABLE
1060 PI=&H1800    :'PATTERN ID TABLE
1070 '
1080 SCREEN 2
```

```
1090 ′
1100 RESTORE 1310
1110 I=PG
1120 GOSUB 1250
1130 ′
1140 RESTORE 1340
1150 I=CT
1160 GOSUB 1250
1170 ′
1180 FOR I=1 TO 1000 : NEXT
1190 ′
1200 FOR I=0 TO 255
1210   VPOKE PI+I,0
1220 NEXT
1230 GOTO 1230
1240 ′
1250 READ A$
1260   IF A$="*" THEN RETURN
1270   VPOKE I,VAL("&H"+A$)
1280   I=I+1
1290 GOTO 1250
1300 ′
1310 DATA 7E,DE,BF,BF,BF,FD,FB,7E
1320 DATA *
1330 ′
1340 DATA D1,DF,DF,DF,DF,D1,D1,D1
1350 DATA *
```

リスト2-2

```
1000 ′   PROGRAM    PATTERN ID TABLE TEST
1010 ′   DATE       19-JUN-84
1020 ′
1030 ′   CONSTANTS
1040 PG=&H0      :′PATTERN GENERATOR TABLE
1050 CT=&H2000   :′COLOR TABLE
1060 PI=&H1800   :′PATTERN ID TABLE
1070 ′
1080 SCREEN 2
1090 ′
1091 DEF USR=&H41
1092 A=USR(0)    :′CALL DISSCR
1100 RESTORE 1310
1110 I=PG
1120 GOSUB 1250
1130 ′
1140 RESTORE 1340
1150 I=CT
1160 GOSUB 1250
1170 ′
1180 FOR I=1 TO 1000 : NEXT
1190 ′
```

```
1191 DEF USR=&H44
1192 A=USR(0)    :'CALL ENASCR
1200 FOR I=0 TO 255
1210   VPOKE PI+I,0
1220 NEXT
1230 GOTO 1230
1240 '
1250 READ A$
1260   IF A$="*" THEN RETURN
1270   VPOKE I,VAL("&H"+A$)
1280   I=I+1
1290 GOTO 1250
1300 '
1310 DATA 7E,DE,BF,BF,BF,FD,FB,7E
1320 DATA *
1330 '
1340 DATA D1,DF,DF,DF,DF,D1,D1,D1
1350 DATA *
```

2. VDPの初期化

ＢＩＯＳにはＶＤＰのモード設定，初期化のルーチンが用意されていますが，ここではＶＤＰのプラグラミングを知ってもらうために，別に初期化ルーチンをつくってみました。ただし，ＶＤＰのレジスタに対する読み書きはＢＩＯＳに用意されているものを使用しています。

プログラムはレジスタの働き，セットの仕方を理解してもらうためのサンプルです(**リスト2-3**)。

リスト2-3

```
1000 ';
1010 ';   PROGRAM INITIALIZE VDP
1020 ';   DATE    6-FEB-84
1030 ';
1040 '         ORG   0D000H
1050 ';
1060 ';   BIOS CALL
1070 ';
1080 ';   WRTVDP (0047H)
1090 ';     WRITES TO VDP RESISTER
1100 ';     ENTRY:   C <= REG.#
1110 ';              B <= DATA
1120 ';     RETURN:  NONE
1130 ';     MODIFIES:AF,BC
1140 ';
1150 'WRTVDP  EQU   0047H
1160 ';
1170 ';   INITIALIZE VDP
1180 ';
```

```
1190 'INIVDP   EQU    $
1200 ';
1210 '         LD     A,8
1220 '         LD     C,0
1230 '         LD     HL,VDPDT
1240 'INIVD1:  LD     B,(HL)
1250 '         PUSH   BC
1260 '         PUSH   AF
1270 '         CALL   WRTVDP
1280 '         POP    AF
1290 '         POP    BC
1300 '         INC    C
1310 '         INC    HL
1320 '         CP     C
1330 '         JR     NZ,INIVD1
1340 '         RET
1350 ';
1360 ';   INITIAL DATA
1370 ';
1380 'VDPDT    EQU    $
1390 ';
1400 ';   REG. #0,#1
1410 ';     EV=0
1420 ';     4/16K=1            ; (VRAM=16K)
1430 ';     BLANK=1            ; (DIPLAY=ON)
1440 ';     IE=1               ; (INTERRUPT ENABLE)
1450 ';     M1=0:M2=0:M3=1     ; (GRAPHIC II MODE)
1460 ';     SIZE=1             ; (SPRITE SIZE=16*16)
1470 ';     MAG=0              ; (SPRITE NON MAG.)
1480 ';
1490 '         DEFB   02H
1500 '         DEFB   0E2H
1510 ';   REG. #2
1520 '         DEFB   06H     ; PAT. ID. TABLE=1800H
1530 ';   REG. #3
1540 '         DEFB   0FFH    ; COLOR TABLE=2000H
1550 ';   REG. #4
1560 '         DEFB   03H     ; PAT. GEN. TABLE=0000H
1570 ';   REG. #5
1580 '         DEFB   36H     ; SPRITE ATT. TABLE=1B00H
1590 ';   REG. #6
1600 '         DEFB   07H     ; SPRITE GEN. TABLE=3800H
1610 ';   REG. #7
1620 '         DEFB   07H     ; BACKDROP COLOR=CYAAN
1630 ';
1640 '         END
```

3. キャラクタをスプライトで表示する

スプライトを使って移動するキャラクタをプログラムしてみました（**リスト2-4**）。プログラムではスプライト上に定義したキャラクタが，カーソル・キーを操作することによって上下左右に動きます。

ＶＤＰの特徴であるスプライト機能を実際にどんなものか確認してほしいと思います。スプライトの色は，１つのスプライトにつき１色しか表現できませんが(実際には透明プラス１色)，これは複数のスプライト（４枚まで）を重ね合わせることにより解決できます。

リスト2-4

```
1000 ´;
1010 ´;   PROGRAM  SPRITE
1020 ´;   DATE     15-APR-84
1030 ´;
1040 ´          ORG   0D000H
1050 ´;
1060 ´ENTRY:    JR    START
1070 ´;
1080 ´;   BIOS CALL
1090 ´;
1100 ´;   INIT32 (006FH)
1110 ´;     INITIALIZES SCREEN FOR
1120 ´;     GRAPHIC I MODE
1130 ´;
1140 ´;   WRTVDP (0047H)
1150 ´;     WRITES TO VDP RESISTER
1160 ´;
1170 ´;   WRTVRM (004DH)
1180 ´;     WRITE TO VRAM
1190 ´;
1200 ´;   LDIRVM (005CH)
1210 ´;     MOVES BLOCK OF MEMORY
1220 ´;     FROM VRAM TO MEMORY
1230 ´;
1240 ´;   GTSTCK (00D5H)
1250 ´;     RETURNS THE CURRENT STATUS
1260 ´;     OF JOY STICK
1270 ´;
1280 ´;   BREAKX (00B7H)
1290 ´;     CHECKS THE STATUS OF
1300 ´;     CTRL-STOP KEY
1310 ´;
1320 ´INIT32   EQU   006FH
1330 ´WRTVDP   EQU   0047H
1340 ´WRTVRM   EQU   004DH
1350 ´LDIRVM   EQU   005CH
1360 ´GTSTCK   EQU   00D5H
```

```
1370 'BREAKX   EQU   00B7H
1380 ';
1390 ';   CONSTANT VALUES
1400 ';
1410 'T32ATR   EQU   0F3C3H
1420 'T32PAT   EQU   0F3C5H
1430 ';
1440 ';   MAIN
1450 ';
1460 'START    EQU   $
1470 ';
1480 '         PUSH  AF
1490 '         PUSH  BC
1500 '         PUSH  DE
1510 '         PUSH  HL
1520 '         CALL  INIVDP        ;INITIALISES VDP
1530 '         LD    DE,(ATRTBL)   ;DE=(X,Y)
1540 'LOOP:    PUSH  DE            ;SAVE CURRENT (X,Y)
1550 '         XOR   A
1560 '         CALL  GTSTCK        ;GET CURSOR KEY
1570 '         POP   DE            ;DE=CURRENT (X,Y)
1580 '         CP    1             ;UP?
1590 '         JR    NZ,L1
1600 '         DEC   D             ;YES,Y=Y-1
1610 'L1:      CP    3             ;RIGHT?
1620 '         JR    NZ,L2
1630 '         INC   E             ;YES,X=X+1
1640 'L2:      CP    5             ;DOWN?
1650 '         JR    NZ,L3
1660 '         INC   D             ;YES,Y=Y+1
1670 'L3:      CP    7             ;LEFT?
1680 '         JR    NZ,L4
1690 '         DEC   E             ;YES,X=X-1
1700 'L4:      LD    HL,(T32ATR)
1710 '         LD    A,D
1720 '         CALL  WRTVRM        ;SET Y
1730 '         INC   HL
1740 '         LD    A,E
1750 '         CALL  WRTVRM        ;SET X
1760 '         LD    BC,800H       ;DELAY
1770 'DELAY:   DEC   BC
1780 '         LD    A,C
1790 '         OR    B             ;COMPLETE?
1800 '         JR    NZ,DELAY      ;NO,CONTINUE
1810 '         CALL  BREAKX        ;CHECK CTRL-STOP
1820 '         JR    NC,LOOP
1830 '         POP   HL
1840 '         POP   DE
```

```
1850 '        POP    BC
1860 '        POP    AF
1870 '        RET
1880 ';
1890 ';   INITIALISES
1900 ';
1910 'INIVDP   EQU    $
1920 ';
1930 '        CALL   INIT32        ;INITIALIZE SCREEN
1940 '        LD     BC,0E201H     ;SPRITE SIZE=16*16
1950 '        CALL   WRTVDP        ;WRITE TO REGISTER #1
1960 '        LD     BC,4          ;INIT SPRITE ATR.
1970 '        LD     DE,(T32ATR)
1980 '        LD     HL,ATRTBL
1990 '        CALL   LDIRVM        ;TRANSFER MEMORY TO VRAM
2000 '        LD     BC,32         ;INIT SPRITE #0
2010 '        LD     DE,(T32PAT)
2020 '        LD     HL,PATTBL
2030 '        CALL   LDIRVM        ;TRANSFER MEMORY TO VRAM
2040 '        RET
2050 ';
2060 ';   INITIAL DATA OF ATR.
2070 ';
2080 'ATRTBL  EQU    $
2090 ';
2100 '        DEFB   0FFH          ;Y
2110 '        DEFB   0H            ;X
2120 '        DEFB   0H            ;SPRITE #0
2130 '        DEFB   08H           ;COLOR=8
2140 ';
2150 ';   PATTERN DATA
2160 ';
2170 'PATTBL  EQU    $
2180 ';
2190 '        DEFB   000H,001H,01DH,07FH
2200 '        DEFB   07BH,0FCH,0FFH,0FFH
2210 '        DEFB   0FFH,0FFH,0FFH,07FH
2220 '        DEFB   07FH,03FH,03FH,01EH
2230 '        DEFB   080H,000H,01CH,07EH
2240 '        DEFB   0FFH,0FFH,0FFH,0FDH
2250 '        DEFB   0FDH,0FBH,0FBH,0FAH
2260 '        DEFB   0FEH,0FCH,0FCH,070H
2270 ';
2280 '        END
```

4. 重ね合わせの処理

重ね合わせの処理とは，背景画面と移動キャラクタが重なった場合の処理を指します。背景画面にキャラクタ・パターン面，移動キャラクタをスプライトで表示する，といった場合は重ね合わせをまったく意識する必要はありません。

しかし，スプライトの表示制限を考えると移動キャラクタをすべてスプライトにするといったことが不可能になる場合があります。そこで移動キャラクタをキャラクタ・パターン面に表示しなければならなくなった場合，当然背景画面と同じ面に表示するので，背景に影響を与えないようにしなければなりません。

スプライトの移動であれば，スプライト属性テーブルの座標を書き変えるだけで済みますが，キャラクタ・パターン面での移動は，現在表示しているものを消し，移動後の位置に表示する，といった作業にする必要があります。これは，アニメの原理と同じです。

さて重ね合わせの処理ですが，キャラクタを移動するときに「現在表示しているものを消し」という作業を「前に表示していた背景を表示する」という作業にすると背景画面に影響は与えません。**リスト2-5**は，その例です。

リスト2-5

```
1000 '; PROGRAM  SUPERIMPOSE
1010 '; DATE     19-JUN-84
1020 ';
1030 ';
1040 '  ORG     0C000H
1050 ';
1060 ';  BIOS CALL
1070 ';
1080 ';DISSCR   (0041H)
1090 ';  DISPLAY ON
1100 ';  ENTRY:   NONE
1110 ';  RETURN:  NONE
1120 ';  MODIFIES:AF,BC
1130 ';ENASCR   (0044H)
1140 ';  DISPLAY OFF
1150 ';  ENTRY:   NONE
1160 ';  RETURN:  NONE
1170 ';  MODIFIES:AF,BC
1180 ';CHGMOD   (005FH)
1190 ';  CHANGE SCREEN MODE
1200 ';  ENTRY:   SCRMOD(FCAFH)
1210 ';  RETURN:  NONE
1220 ';  MODIFIES:ALL
1230 ';CHGCLR   (0062H)
1240 ';  CHANGE COLOR
```

```
1250 ';  ENTRY:    FORCLR(F3E9H)
1260 ';            BAKCLR(F3EAH)
1270 ';            BDRCLR(F3EBH)
1280 ';  RETURN:   NONE
1290 ';  MODIFIES:ALL
1300 ';WRTVRM   (004DH)
1310 ';  WRITES TO VRAM
1320 ';  ENTRY:    VRAM ADR => HL
1330 ';            DATA     => A
1340 ';  RETURN:   NONE
1350 ';  MODIFIES:AF
1360 ';RDVRM    (004AH)
1370 ';  READS FROM VRAM
1380 ';  ENTRY:    VRAM ADR => HL
1390 ';  RETURN:   A
1400 ';  MODIFIES:AF
1410 ';GTSTCK   (00D5H)
1420 ';  GET DIRECTION OF STICK
1430 ';  ENTRY:    STICK No. => A
1440 ';  RETURN:   DIRECTION OF STICK => A
1450 ';  MODIFIES:ALL
1460 ';BREAKX   (00B7H)
1470 ';  TEST CTRL-STOP KEY
1480 ';  ENTRY:    NONE
1490 ';  RETURN:   CARRY FLAG
1500 ';  MODIFIES:AF
1510 ';CLS      (00D5H)
1520 ';  CLEAR SCREEN
1530 ';  ENTRY:    ZERO FLAG ON
1540 ';  RETURN:   NONE
1550 ';  MODIFIES:AF,BC,DE
1560 ';
1570 'DISSCR    EQU     0041H
1580 'ENASCR    EQU     0044H
1590 'CHGMOD    EQU     005FH
1600 'SCRMOD    EQU     0FCAFH
1610 'CHGCLR    EQU     0062H
1620 'FORCLR    EQU     0F3E9H
1630 'BAKCLR    EQU     0F3EAH
1640 'BDRCLR    EQU     0F3EBH
1650 'WRTVRM    EQU     004DH
1660 'RDVRM     EQU     004AH
1670 'CLS       EQU     00C3H
1680 'GTSTCK    EQU     00D5H
1690 'BREAKX    EQU     00B7H
1700 ';
1710 '; SYSTEM CONSTANTS
1720 ';
1730 'PTNGEN    EQU     0        ; PATTERN GENERATOR TABLE
1740 'CLRTBL    EQU     2000H    ; COLOR TABLE
1750 'PTNID     EQU     1800H    ; PATTERN ID TABLE
1760 ';
```

```
1770 '; INIT
1780 ';
1790 'INIT         EQU     $
1800 '             CALL    DISSCR
1810 '             LD      A,2
1820 '             LD      (SCRMOD),A
1830 '             CALL    CHGMOD
1840 '             XOR     A
1850 '             LD      (FORCLR),A
1860 '             LD      (BAKCLR),A
1870 '             LD      (BDRCLR),A
1880 '             CALL    CHGCLR
1890 '             XOR     A       ; SET ZERO FLAG
1900 '             CALL    CLS
1910 '             CALL    DSPINI
1920 '             CALL    DISPLY
1930 '             CALL    ENASCR
1940 ';
1950 '; MAIN LOOP
1960 ';
1970 'MAIN         EQU     $
1980 '             LD      BC,2000H; SET DELAY COUNTS
1990 'DELAY:       DEC     BC
2000 '             LD      A,B
2010 '             OR      C       ; BC=0 ?
2020 '             JR      NZ,DELAY
2030 ';
2040 '             XOR     A       ; STICK ID=0 (KEY_BOARD)
2050 '             CALL    GTSTCK
2060 '             CP      1
2070 '             JP      Z,UP
2080 '             CP      7
2090 '             JP      Z,LEFT
2100 '             CP      3
2110 '             JP      Z,RIGHT
2120 '             CP      5
2130 '             JP      Z,DOWN
2140 '             CALL    BREAKX
2150 '             JR      NC,MAIN
2160 ';
2170 '; RETURN
2180 ';
2190 '             CALL    CLS
2200 '             LD      A,15
2210 '             LD      (FORCLR),A
2220 '             CALL    CHGCLR
2230 '             RET
2240 ';
2250 '; MOVE PATTERN
2260 ';
2270 'UP           EQU     $
2280 '             LD      A,(YPOS)
```

```
2290 '              OR      A        ; YPOS=0?
2300 '              JP      Z,MAIN
2310 '              CALL    ERASE
2320 '              LD      A,(YPOS)
2330 '              DEC     A
2340 '              LD      (YPOS),A
2350 '              CALL    DISPLY
2360 '              JP      MAIN
2370 ';
2380 'LEFT          EQU     $
2390 '              LD      A,(XPOS)
2400 '              OR      A
2410 '              JP      Z,MAIN
2420 '              CALL    ERASE
2430 '              LD      A,(XPOS)
2440 '              DEC     A
2450 '              LD      (XPOS),A
2460 '              CALL    DISPLY
2470 '              JP      MAIN
2480 ';
2490 'RIGHT         EQU     $
2500 '              LD      A,(XPOS)
2510 '              CP      31
2520 '              JP      Z,MAIN
2530 '              CALL    ERASE
2540 '              LD      A,(XPOS)
2550 '              INC     A
2560 '              LD      (XPOS),A
2570 '              CALL    DISPLY
2580 '              JP      MAIN
2590 ';
2600 'DOWN          EQU     $
2610 '              LD      A,(YPOS)
2620 '              CP      7
2630 '              JP      Z,MAIN
2640 '              CALL    ERASE
2650 '              LD      A,(YPOS)
2660 '              INC     A
2670 '              LD      (YPOS),A
2680 '              CALL    DISPLY
2690 '              JP      MAIN
2700 ';
2710 'YPOS:         DEFB    1
2720 'XPOS:         DEFB    1
2730 ';
2740 '; ERASE
2750 ';    (ERASE PATTERN)
2760 'ERASE         EQU     $
2770 '              CALL    PRTADR
2780 '              LD      A,(BACK)
2790 '              CALL    WRTVRM
2800 '              RET
```

```
2810 ';
2820 '; DISPLY
2830 ';    (DISPLAY PATTERN)
2840 'DISPLY     EQU      $
2850 '           CALL     PRTADR
2860 '           CALL     RDVRM
2870 '           LD       (BACK),A
2880 '           LD       A,2
2890 '           CALL     WRTVRM
2900 '           RET
2910 ';
2920 'BACK:      DEFS     1
2930 ';
2940 '; PRTADR
2950 ';    (COMPUTE VRAM ADDRESS)
2960 'PRTADR     EQU      $
2970 '           LD       BC,32
2980 '           LD       HL,PTNID
2990 '           LD       DE,(YPOS)
3000 '           INC      E
3010 'ADRY:      DEC      E
3020 '           JR       Z,ADRX
3030 '           ADD      HL,BC
3040 '           JR       ADRY
3050 'ADRX:      LD       B,0
3060 '           LD       C,D
3070 '           ADD      HL,BC
3080 '           RET
3090 ';
3100 '; DSPINI
3110 ';    (DISPLAY INITIAL SCREEN)
3120 'DSPINI     EQU      $
3130 '           LD       HL,PTNGEN
3140 '           LD       DE,PTNDAT
3150 '           LD       B,PTNDT1-PTNDAT
3160 '           CALL     DATXFR
3170 ';
3180 '           LD       HL,CLRTBL
3190 '           LD       DE,CLRDAT
3200 '           LD       B,CLRDT1-CLRDAT
3210 '           CALL     DATXFR
3220 ';
3230 '           LD       HL,PTNID
3240 '           LD       C,4
3250 'LOOP1:     LD       B,32
3260 'LOOP2:     XOR      A
3270 '           CALL     WRTVRM
3280 '           INC      HL
3290 '           DJNZ     LOOP2
3300 '           LD       B,32
3310 'LOOP3:     LD       A,1
3320 '           CALL     WRTVRM
```

```
3330 '              INC      HL
3340 '              DJNZ     LOOP3
3350 '              DEC      C
3360 '              JR       NZ,LOOP1
3370 '              RET
3380 ';
3390 '; DATXFR
3400 ';    DATA => VRAM
3410 'DATXFR        EQU      $
3420 'LOOP:         LD       A,(DE)
3430 '              CALL     WRTVRM
3440 '              INC      HL
3450 '              INC      DE
3460 '              DJNZ     LOOP
3470 '              RET
3480 ';
3490 '; PATTERN & COLOR DATA
3500 ';
3510 'PTNDAT        EQU      $
3520 '              DEFB     7EH,0DEH,0BFH,0BFH
3530 '              DEFB     0BFH,0FDH,0FBH,7EH
3540 '              DEFB     3CH,7EH,0DFH,0BFH
3550 '              DEFB     0BFH,0FBH,7EH,3CH
3560 '              DEFB     3CH,3CH,99H,0FFH
3570 '              DEFB     18H,18H,24H,0C3H
3580 'PTNDT1        EQU      $
3590 ';
3600 'CLRDAT        EQU      $
3610 '              DEFB     0D1H,0DFH,0DFH,0DFH
3620 '              DEFB     0DFH,0D1H,0D1H,0D1H
3630 '              DEFB     71H,71H,7FH,7FH
3640 '              DEFB     7FH,71H,71H,71H
3650 '              DEFB     0B1H,0B1H,0B1H,0B1H
3660 '              DEFB     0B1H,0B1H,0B1H,0B1H
3670 'CLRDT1        EQU      $
```

5. 仮想VRAM

仮想ＶＲＡＭというのは，画面処理のためのひとつの手法です。たとえばパターンを移動するときにその方向に何があるかという判断をＶＲＡＭから読み込んで行うのではなく，メイン・メモリ上に取った仮想ＶＲＡＭを参照する，といった場合です。つまり，仮想ＶＲＡＭとは，実際に表示しているものをコード化してメイン・メモリ上に置き，画面の処理の際に参照されるものです。

この手法は，迷路タイプのゲームには必ずといってよいほど使われています。

リスト2-6 に仮想ＶＲＡＭの例を示します。リストで VIRTULというラベルのついたものが仮想ＶＲＡＭのデータです。このデータは

パターンの名称番号そのものです。

仮想ＶＲＡＭのデータを，たとえば壁があるところとないところという区別にすれば，これは１ビットで表わすことができます。こういったデータの圧縮を行うと，ゲームの画面数を増やすことができます。

リスト2-6

```
1000 '; PROGRAM  VIRTUAL VRAM
1010 '; DATE     19-JUN-84
1020 ';
1030 ';
1040 '  ORG     0C000H
1050 ';
1060 ';  BIOS CALL
1070 ';
1080 ';DISSCR   (0041H)
1090 ';  DISPLAY ON
1100 ';  ENTRY:   NONE
1110 ';  RETURN:  NONE
1120 ';  MODIFIES:AF,BC
1130 ';ENASCR   (0044H)
1140 ';  DISPLAY OFF
1150 ';  ENTRY:   NONE
1160 ';  RETURN:  NONE
1170 ';  MODIFIES:AF,BC
1180 ';CHGMOD   (005FH)
1190 ';  CHANGE SCREEN MODE
1200 ';  ENTRY:   SCRMOD(FCAFH)
1210 ';  RETURN:  NONE
1220 ';  MODIFIES:ALL
1230 ';CHGCLR   (0062H)
1240 ';  CHANGE COLOR
1250 ';  ENTRY:   FORCLR(F3E9H)
1260 ';           BAKCLR(F3EAH)
1270 ';           BDRCLR(F3EBH)
1280 ';  RETURN:  NONE
1290 ';  MODIFIES:ALL
1300 ';WRTVRM   (004DH)
1310 ';  WRITES TO VRAM
1320 ';  ENTRY:   VRAM ADR => HL
1330 ';           DATA     => A
1340 ';  RETURN:  NONE
1350 ';  MODIFIES:A
1360 ';BEEP     (00C0H)
1370 ';  BEEP ON
1380 ';  ENTRY:   NONE
1390 ';  RETURN:  NONE
1400 ';  MODIFIES:ALL
1410 ';GTSTCK   (00D5H)
1420 ';  GET DIRECTION OF STICK
1430 ';  ENTRY:   STICK No. => A
```

```
1440 ';  RETURN:  DIRECTION OF STICK => A
1450 ';  MODIFIES:ALL
1460 ';BREAKX   (00B7H)
1470 ';  TEST CTRL-STOP KEY
1480 ';  ENTRY:   NONE
1490 ';  RETURN:  CARRY FLAG
1500 ';  MODIFIES:AF
1510 ';CLS      (00D5H)
1520 ';  CLEAR SCREEN
1530 ';  ENTRY:   ZERO FLAG ON
1540 ';  RETURN:  NONE
1550 ';  MODIFIES:AF,BC,DE
1560 ';
1570 'DISSCR    EQU     0041H
1580 'ENASCR    EQU     0044H
1590 'CHGMOD    EQU     005FH
1600 'SCRMOD    EQU     0FCAFH
1610 'CHGCLR    EQU     0062H
1620 'FORCLR    EQU     0F3E9H
1630 'BAKCLR    EQU     0F3EAH
1640 'BDRCLR    EQU     0F3EBH
1650 'WRTVRM    EQU     004DH
1660 'BEEP      EQU     00C0H
1670 'CLS       EQU     00C3H
1680 'GTSTCK    EQU     00D5H
1690 'BREAKX    EQU     00B7H
1700 ';
1710 '; SYSTEM CONSTANTS
1720 ';
1730 'PTNGEN    EQU     0        ; PATTERN GENERATOR TABLE
1740 'CLRTBL    EQU     2000H    ; COLOR TABLE
1750 'PTNID     EQU     1800H    ; PATTERN ID TABLE
1760 ';
1770 '; INIT
1780 ';
1790 'INIT      EQU     $
1800 '          CALL    DISSCR
1810 '          LD      A,2
1820 '          LD      (SCRMOD),A
1830 '          CALL    CHGMOD
1840 '          XOR     A
1850 '          LD      (FORCLR),A
1860 '          LD      (BAKCLR),A
1870 '          LD      (BDRCLR),A
1880 '          CALL    CHGCLR
1890 '          XOR     A        ; SET ZERO FLAG
1900 '          CALL    CLS
1910 '          CALL    DSPINI
1920 '          CALL    DISPLY
1930 '          CALL    ENASCR
1940 ';
1950 '; MAIN LOOP
```

```
1960 ';
1970 'MAIN        EQU     $
1980 '            LD      BC,2000H; SET DELAY COUNTS
1990 'DELAY:      DEC     BC
2000 '            LD      A,B
2010 '            OR      C        ; BC=0 ?
2020 '            JR      NZ,DELAY
2030 ';
2040 '            XOR     A        ; STICK ID=0 (KEY_BOARD)
2050 '            CALL    GTSTCK
2060 '            CP      1
2070 '            JP      Z,UP
2080 '            CP      7
2090 '            JP      Z,LEFT
2100 '            CP      3
2110 '            JP      Z,RIGHT
2120 '            CP      5
2130 '            JP      Z,DOWN
2140 '            CALL    BREAKX
2150 '            JR      NC,MAIN
2160 ';
2170 '; RETURN
2180 ';
2190 '            CALL    CLS
2200 '            LD      A,15
2210 '            LD      (FORCLR),A
2220 '            CALL    CHGCLR
2230 '            RET
2240 ';
2250 '; MOVE PATTERN
2260 ';
2270 'UP          EQU     $
2280 '            LD      DE,(YPOS) ; D=XPOS:E=YPOS
2290 '            DEC     E
2300 '            CALL    VDATA
2310 '            OR      A        ; WALL?
2320 '            JP      Z,MAIN
2330 '            CALL    ERASE
2340 '            LD      A,(YPOS)
2350 '            DEC     A
2360 '            LD      (YPOS),A
2370 '            JP      NEXT
2380 ';
2390 'LEFT        EQU     $
2400 '            LD      DE,(YPOS)
2410 '            DEC     D
2420 '            CALL    VDATA
2430 '            OR      A
2440 '            JP      Z,MAIN
2450 '            CALL    ERASE
2460 '            LD      A,(XPOS)
2470 '            DEC     A
```

```
2480 '          LD      (XPOS),A
2490 '          JP      NEXT
2500 ';
2510 'RIGHT     EQU     $
2520 '          LD      DE,(YPOS)
2530 '          INC     D
2540 '          CALL    VDATA
2550 '          OR      A
2560 '          JP      Z,MAIN
2570 '          CALL    ERASE
2580 '          LD      A,(XPOS)
2590 '          INC     A
2600 '          LD      (XPOS),A
2610 '          JP      NEXT
2620 ';
2630 'DOWN      EQU     $
2640 '          LD      DE,(YPOS)
2650 '          INC     E
2660 '          CALL    VDATA
2670 '          OR      A
2680 '          JP      Z,MAIN
2690 '          CALL    ERASE
2700 '          LD      A,(YPOS)
2710 '          INC     A
2720 '          LD      (YPOS),A
2730 ';
2740 'NEXT      EQU     $
2750 '          CALL    DISPLY
2760 '          LD      A,(VIRTDT)
2770 '          CP      1           ; BALL?
2780 '          CALL    Z,BELRK
2790 '          JP      MAIN
2800 ';
2810 'YPOS:     DEFB    1
2820 'XPOS:     DEFB    1
2830 ';
2840 '; FETCH VIRTUAL VRAM DATA
2850 ';    (HL=VIRTUL+YPOS*8+XPOS)
2860 'VDATA     EQU     $
2870 '          LD      BC,8
2880 '          LD      HL,VIRTUL
2890 '          INC     E
2900 'VIRTY:    DEC     E
2910 '          JR      Z,VIRTX
2920 '          ADD     HL,BC
2930 '          JR      VIRTY
2940 'VIRTX:    LD      B,0
2950 '          LD      C,D
2960 '          ADD     HL,BC
2970 '          LD      (VIRTNO),HL
2980 '          LD      A,(HL)
2990 '          LD      (VIRTDT),A
```

```
3000 '             RET
3010 ';
3020 'VIRTNO:      DEFS      2
3030 'VIRTDT:      DEFS      1
3040 ';
3050 '; BEEP ON AND DELETE BALL DATA
3060 ';
3070 'BELRK        EQU       $
3080 '             CALL      BEEP
3090 '             LD        HL,(VIRTNO)
3100 '             LD        (HL),3    ; CLEAR BALL DATA
                                         ON VIRTUAL VRAM
3110 '             RET
3120 ';
3130 '; ERASE
3140 ';    (ERASE PATTERN)
3150 'ERASE        EQU       $
3160 '             CALL      PRTADR
3170 '             LD        A,3
3180 '             CALL      WRTVRM
3190 '             RET
3200 ';
3210 '; DISPLY
3220 ';    (DISPLAY PATTERN)
3230 'DISPLY       EQU       $
3240 '             CALL      PRTADR
3250 '             LD        A,2
3260 '             CALL      WRTVRM
3270 '             RET
3280 ';
3290 '; PRTADR
3300 ';    (COMPUTE VRAM ADDRESS)
3310 'PRTADR       EQU       $
3320 '             LD        BC,32
3330 '             LD        HL,PTNID
3340 '             LD        DE,(YPOS)
3350 '             INC       E
3360 'ADRY:        DEC       E
3370 '             JR        Z,ADRX
3380 '             ADD       HL,BC
3390 '             JR        ADRY
3400 'ADRX:        LD        B,0
3410 '             LD        C,D
3420 '             ADD       HL,BC
3430 '             RET
3440 ';
3450 '; DSPINI
3460 ';    (DISPLAY INITIAL SCREEN)
3470 'DSPINI       EQU       $
3480 '             LD        HL,PTNGEN
3490 '             LD        DE,PTNDAT
3500 '             LD        B,PTNDT1-PTNDAT
```

```
3510 '          CALL    DATXFR
3520 ';
3530 '          LD      HL,CLRTBL
3540 '          LD      DE,CLRDAT
3550 '          LD      B,CLRDT1-CLRDAT
3560 '          CALL    DATXFR
3570 ';
3580 '          LD      HL,PTNID
3590 '          LD      DE,VIRTUL
3600 '          LD      C,8
3610 'LOOP1:    LD      B,8
3620 '          CALL    DATXFR
3630 '          DEC     C
3640 '          RET     Z
3650 '          PUSH    DE
3660 '          LD      DE,24
3670 '          ADD     HL,DE    ; RENEW PTN ID TABLE
3680 '          POP     DE
3690 '          JR      LOOP1
3700 ';
3710 '; DATXFR
3720 ';    DATA => VRAM
3730 'DATXFR    EQU     $
3740 'LOOP2:    LD      A,(DE)
3750 '          CALL    WRTVRM
3760 '          INC     HL
3770 '          INC     DE
3780 '          DJNZ    LOOP2
3790 '          RET
3800 ';
3810 '; PATTERN & COLOR DATA
3820 ';
3830 'PTNDAT    EQU     $
3840 '          DEFB    7EH,0DEH,0BFH,0BFH
3850 '          DEFB    0BFH,0FDH,0FBH,7EH
3860 '          DEFB    3CH,7EH,0DFH,0BFH
3870 '          DEFB    0BFH,0FBH,7EH,3CH
3880 '          DEFB    3CH,3CH,99H,0FFH
3890 '          DEFB    18H,18H,24H,0C3H
3900 'PTNDT1    EQU     $
3910 ';
3920 'CLRDAT    EQU     $
3930 '          DEFB    0D1H,0DFH,0DFH,0DFH
3940 '          DEFB    0DFH,0D1H,0D1H,0D1H
3950 '          DEFB    71H,71H,7FH,7FH
3960 '          DEFB    7FH,71H,71H,71H
3970 '          DEFB    0B1H,0B1H,0B1H,0B1H
3980 '          DEFB    0B1H,0B1H,0B1H,0B1H
3990 'CLRDT1    EQU     $
4000 ';
4010 '; VIRTUAL VRAM DATA
4020 ';
```

```
4030 'VIRTUL    EQU     $
4040 '          DEFB    0,0,0,0,0,0,0,0
4050 '          DEFB    0,3,3,3,3,3,3,0
4060 '          DEFB    0,3,0,0,0,1,3,0
4070 '          DEFB    0,3,3,1,3,3,3,0
4080 '          DEFB    0,3,0,0,3,0,1,0
4090 '          DEFB    0,3,0,3,3,0,3,0
4100 '          DEFB    0,3,3,1,3,3,3,0
4110 '          DEFB    0,0,0,0,0,0,0,0
```

6. スクロール

ＶＤＰではパターン名称テーブルをいじることにより，高速にスクロールを行うことができます(**リスト2-7**)。

サブルーチンFRVRMはVRAMの名称テーブルをWORKから始まるアドレスへ転送します。そして，WORKの内容を右または左へスクロールして，サブルーチンTOVRMでVRAMに書き込みます。

ＷＯＲＫの内容を右へスクロールするには画面の１行ごとに**図2-37**のようにＨＬ，ＤＥをセットして，右端のデータをＡレジスタに保存し，ＬＤＤＲ命令を実行します。実行後，ＨＬ，ＤＥは**図2-38**のアドレスを示しますから，ＡレジスタをＤＥの示すアドレスへいれます。これを24回繰り返すと一画面が右へスクロールします。左へスクロールするときはＬＤＩＲ命令を使います。

リスト2-8 はテスト・プログラムで，ＲＵＮした後，カーソルの→を押すと右へ，←へ押すと左へスクロールします。

図2-37

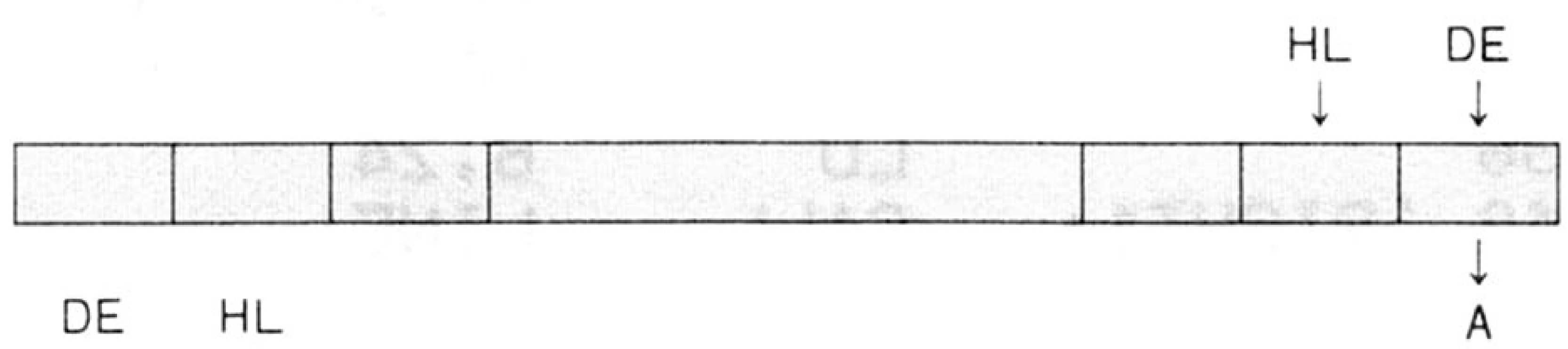
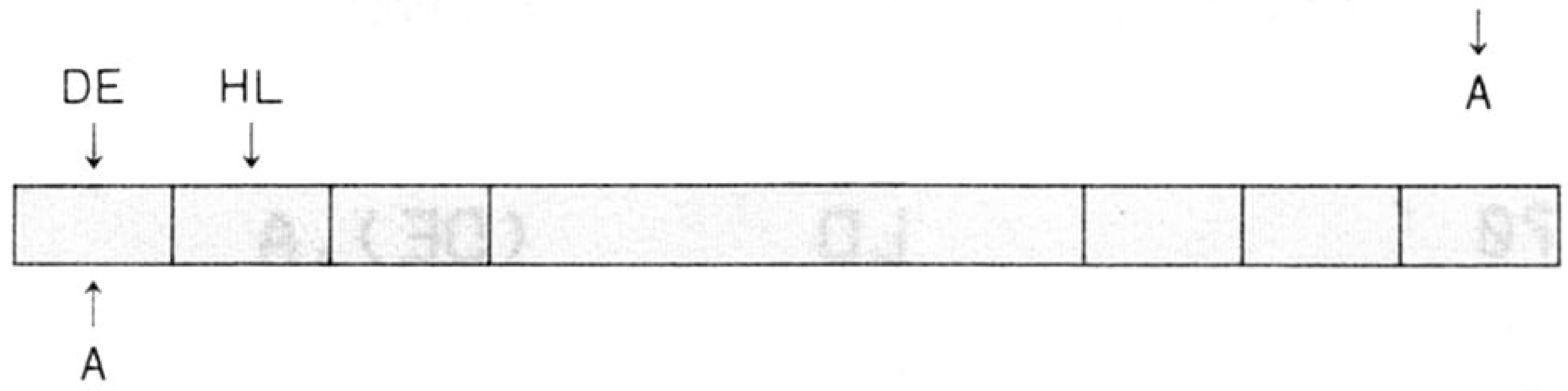


図2-38



リスト2-7

```
1000 ';PROGRAM   SCROLL
1010 ';DATE 29TH JUNE 84
1020 ';
1030 ';
1040 '    ORG       0C000H
1050 ';
1060 ';   BIOS      CALL
1070 ';
1080 ';LDIRMV     (0059H)
1090 ';   MOVE DATA FROM VRAM TO MEMORY
1100 ';   ENTRY:    VRAM ADR    => HL
```

```
1110 ';             MEMORY ADR => DE
1120 ';             LENGTH     => BC
1130 ';   RETURN:   NONE
1140 ';   MODIFIES:ALL
1150 ';LDIRVM    (005CH)
1160 ';   MOVE DATA FROM MEMORY TO VRAM
1170 ';   ENTRY:    MEMORY ADR => HL
1180 ';             VRAM ADR   => DE
1190 ';             LENGTH     => BC
1200 ';   RETURN:   NONE
1210 ';   MODIFIES:ALL
1220 ';
1230 'LDIRMV    EQU     0059H
1240 'LDIRVM    EQU     005CH
1250 '
1260 ';
1270 ';RIGHT
1280 ';
1290 'RIGHT     EQU     $
1300 '          JR      RIGHT0
1310 ';
1320 ';LEFT
1330 ';
1340 'LEFT      EQU     $
1350 '          JR      LEFT0
1360 ';
1370 ';RIGHT0
1380 ';
1390 'RIGHT0    EQU     $
1400 '          CALL    FRVRM     ;READ VRAM
1410 '          LD      HL,WORK-1
1420 '          LD      (DEST),HL
1430 '          LD      B,24
1440 'RIGHT1:   CALL    LINE
1450 '          DEC     HL
1460 '          LDDR              ;MOVE TO RIGHT
1470 '          LD      (DE),A
1480 '          EXX
1490 '          DJNZ    RIGHT1
1500 '          CALL    TOVRM     ;WRITE VRAM
1510 '          RET
1520 ';
1530 ';LEFT0
1540 ';
1550 'LEFT0     EQU     $
1560 '          CALL    FRVRM     ;READ VRAM
1570 '          LD      HL,WORK-32
1580 '          LD      (DEST),HL
1590 '          LD      B,24
1600 'LEFT1:    CALL    LINE
1610 '          INC     HL
1620 '          LDIR              ;MOVE TO LEFT
```

```
1630 '          LD     (DE),A
1640 '          EXX
1650 '          DJNZ   LEFT1
1660 '          CALL   TOVRM     ;WRITE VRAM
1670 '          RET
1680 ';
1690 ';LINE
1700 ';
1710 'LINE      EQU    $
1720 '          EXX
1730 '          LD     BC,31
1740 '          LD     HL,(DEST)
1750 '          LD     DE,32
1760 '          ADD    HL,DE
1770 '          LD     (DEST),HL
1780 '          LD     E,L
1790 '          LD     D,H
1800 '          LD     A,(DE)
1810 '          RET
1820 ';
1830 ';FRVRM
1840 ';
1850 'FRVRM     EQU    $
1860 '          LD     HL,1800H ;SET PATTERN ID TABLE
1870 '          LD     DE,WORK
1880 '          LD     BC,0300H ;SET LENGTH OF WORK
1890 '          CALL   LDIRMV
1900 '          RET
1910 ';
1920 ';TOVRM
1930 ';
1940 'TOVRM     EQU    $
1950 '          LD     HL,WORK
1960 '          LD     DE,1800H ;SET PATTERN ID TABLE
1970 '          LD     BC,0300H
1980 '          CALL   LDIRVM    ;SET WORK LENGTH OF WORK
1990 '          RET
2000 ';
2010 ';VAR
2020 ';
2030 'DEST:     DEFS   2
2040 'WORK:     DEFS   300H
```

リスト2-8

```
10 DEF USR0=&HC000:DEF USR1=&HC002
20 SCREEN 2
30 LINE(0,0)-(255,191),15
40 LINE(255,0)-(0,191),15
50 IF STICK(0)=3 THEN A=USR0(0):GOTO 70
60 IF STICK(0)=7 THEN A=USR1(0)
70 GOTO 50
```

7. 点を打つ，線を引く，円を描く

ここではマシン語で，点を打つ，線を引く，円を描くプログラム(リスト2-9) を紹介します。

リスト2-9

```
; PROGRAM LINE & CIRCLE
; DATE 20TH JUNE 84
;
;
;
  ORG      0C000H
;
;  BIOS CALL
;
;RDVRM    (004AH)
;  READ VRAM
;  ENTRY:   VRAM ADR => HL
;  RETURN:  DATA     => A
;  MODIFIES:AF
;WRTVRM   (004DH)
;  WRITE VRAM
;  ENTRY:   VRAM ADR => HL
;           DATA     => A
;  RETURN:  NONE
;  MODIFIES:AF
RDVRM      EQU      004AH
WRTVRM     EQU      004DH
;
; PSET
;
PSET       EQU      $
           JP       PSTSTR
;
; LINE
;
LINE       EQU      $
           JP       LINSTR
;
; CIRCLE
;
CIRCLE     EQU      $
           JP       CIRSTR
;
; PSTSTR
;
PSTSTR     EQU      $
           LD       BC,(X0)
PSTENT:    LD       A,C
           LD       (PSTX),A
```

```
1440 '          LD     A,B
1450 '          CP     192      ;CHECK Y LIMIT
1460 '          JR     NC,PSTEXT
1470 '          SRL    A        ;CALC VRAM ADDRESS
1480 '          SRL    A
1490 '          SRL    A
1500 '          LD     D,A
1510 '          OR     20H
1520 '          LD     H,A
1530 '          LD     A,C
1540 '          AND    0F8H
1550 '          LD     L,A
1560 '          LD     A,B
1570 '          AND    07H
1580 '          ADD    A,L
1590 '          LD     L,A
1600 '          LD     E,A
1610 '          LD     (PATADR),DE
1620 '          LD     (COLADR),HL
1630 '          CALL   RDVRM
1640 '          LD     C,A      ;C := PRESENT COLOR
1650 '          EX     DE,HL
1660 '          CALL   RDVRM
1670 '          LD     B,A      ;B := PRESENT PATTERN
1680 '          LD     A,(COLOR)
1690 '          LD     E,A
1700 '          LD     A,C
1710 '          AND    0F7H
1720 '          CP     E
1730 '          JR     Z,PSET1
1740 '          LD     C,A
1750 '          LD     A,E
1760 '          ADD    A,A
1770 '          ADD    A,A
1780 '          ADD    A,A
1790 '          ADD    A,A
1800 '          OR     C        ;A := NEW COLOR
1810 '          LD     HL,(COLADR)
1820 '          CALL   WRTVRM
1830 '          LD     HL,TBL1
1840 '          CALL   GETTBL
1850 '          OR     (HL)     ;SET DOT
1860 '          JR     PSET2
1870 'PSET1:    LD     HL,TBL2
1880 '          CALL   GETTBL
1890 '          AND    (HL)     ;RESET DOT
1900 'PSET2:    LD     HL,(PATADR)
1910 '          CALL   WRTVRM
1920 'PSTEXT:   RET
```

```
1930 ';
1940 ';GETTBL
1950 ';
1960 'GETTBL      EQU      $
1970 '            LD       A,(PSTX)
1980 '            AND      07H
1990 '            LD       C,A
2000 '            LD       A,B
2010 '            LD       B,0
2020 '            ADD      HL,BC    ;HL := DATA ADDRESS
2030 '            RET
2040 '
2050 'TBL1:       DEFB     80H,40H,20H
2060 '            DEFB     10H,08H,04H
2070 '            DEFB     02H,01H
2080 'TBL2:       DEFB     07FH,0BFH,0DFH
2090 '            DEFB     0EFH,0F7H,0FBH
2100 '            DEFB     0FDH,0FEH
2110 ';
2120 ';LINSTR
2130 ';
2140 'LINSTR      EQU      $
2150 '            XOR      A
2160 '            LD       (FLG),A
2170 '            LD       BC,0101H;B := Y_INC,C := X_INC
2180 '            LD       HL,(X0) ;H := Y0,L := X0
2190 '            LD       A,(Y1)
2200 '            SUB      H
2210 '            JR       NC,L1
2220 '            LD       B,0FFH   ;Y_INC := -1
2230 '            NEG
2240 'L1:         LD       D,A      ;D := Y_DIFFER
2250 '            LD       A,(X1)
2260 '            SUB      L
2270 '            JR       NC,L2
2280 '            LD       C,0FFH   ;X_INC := -1
2290 '            NEG
2300 'L2:         LD       E,A      ;E := X_DIFFER
2310 '            CP       D
2320 '            JR       NC,L3
2330 '            LD       A,1
2340 '            LD       (FLG),A
2350 '            LD       A,L      ;SWAP X0,Y0
2360 '            LD       L,H
2370 '            LD       H,A
2380 '            LD       (X0),HL
2390 '            LD       HL,(X1) ;SWAP X1,Y1
2400 '            LD       A,L
2410 '            LD       L,H
2420 '            LD       H,A
```

```
2430 '          LD      (X1),HL
2440 '          LD      A,E      ;SWAP X,Y DIFFER
2450 '          LD      E,D
2460 '          LD      D,A
2470 '          LD      A,C      ;SWAP X,Y INC
2480 '          LD      C,B
2490 '          LD      B,A
2500 'L3:       LD      A,E
2510 '          SRL     A
2520 '          LD      (RM),A   ;INIT REMAIN
2530 'L5:       EXX
2540 '          LD      BC,(X0)
2550 '          LD      A,(FLG)
2560 '          OR      A
2570 '          JR      Z,L4
2580 '          LD      A,C      ;SWAP X0,Y0
2590 '          LD      C,B
2600 '          LD      B,A
2610 'L4:       CALL    PSTENT
2620 '          LD      HL,(X1) ;L := X1
2630 '          LD      A,(X0)
2640 '          CP      L
2650 '          JR      Z,LINEXT;EXIT LOOP
2660 '          EXX
2670 '          ADD     A,C
2680 '          LD      (X0),A   ;X0 := X0 + X_INC
2690 '          LD      A,(RM)
2700 '          ADD     A,D
2710 '          LD      (RM),A   ;RM := RM + Y_DIFFER
2720 '          CP      E
2730 '          JR      C,L5
2740 '          SUB     E
2750 '          LD      (RM),A   ;RM := RM - X_DIFFER
2760 '          LD      A,(Y0)
2770 '          ADD     A,B
2780 '          LD      (Y0),A   ;Y0 := Y0 + Y_INC
2790 '          JR      L5
2800 'LINEXT:   RET
2810 ';
2820 ';CIRSTR
2830 ';
2840 'CIRSTR    EQU     $
2850 '          LD      HL,0
2860 '          LD      (X),HL
2870 '          LD      (Y),HL
2880 '          LD      A,(X1)
2890 '          LD      (XX),A   ;SET START POINT
2900 '          LD      A,(Y1)
2910 '          LD      (YY),A
```

```
2920 '          LD      (COUNT),HL
2930 'C1:       LD      A,(X0)
2940 '          LD      C,A
2950 '          LD      A,(XX)
2960 '          ADD     A,C
2970 '          LD      C,A
2980 '          LD      A,(Y0)
2990 '          LD      B,A
3000 '          LD      A,(YY)
3010 '          ADD     A,B
3020 '          LD      B,A
3030 '          CALL    PSTENT
3040 '          LD      HL,(Y)
3050 '          CALL    DIV
3060 '          EX      DE,HL
3070 '          LD      HL,(X)
3080 '          OR      A
3090 '          SBC     HL,DE
3100 '          LD      (X),HL   ;X := X - (Y / 128)
3110 '          CALL    DIV
3120 '          LD      DE,(Y)
3130 '          ADD     HL,DE
3140 '          LD      (Y),HL   ;Y := X / 128 + Y
3150 '          LD      HL,(COUNT)
3160 '          INC     HL
3170 '          LD      (COUNT),HL
3180 '          LD      DE,806
3190 '          OR      A
3200 '          SBC     HL,DE
3210 '          JR      NZ,C1    ;COUNT = 806 ?
3220 '          RET
3230 ';
3240 ';DIV
3250 ';
3260 'DIV       EQU     $
3270 '          LD      B,7      ;HL := HL/128
3280 'DLOOP:    SRA     H
3290 '          RR      L
3300 '          DJNZ    DLOOP
3310 '          RET
3320 '
3330 ';
3340 ';VAR
3350 ';
3360 '
3370 '          ORG     0C200H
3380 'COLOR:    DEFS    1
3390 'X0:       DEFS    1
3400 'Y0:       DEFS    1
```

```
3410 'X1:       DEFS   1
3420 'Y1:       DEFS   1
3430 'FLG:      DEFS   1
3440 'RM:       DEFS   1
3450 'PSTX:     DEFS   1
3460 'PATADR:   DEFS   2
3470 'COLADR:   DEFS   2
3480 'X:        DEFS   1
3490 'XX:       DEFS   1
3500 'Y:        DEFS   1
3510 'YY:       DEFS   1
3520 'COUNT:    DEFS   2
```

●点を打つ

C200H番地にカラー，C201H番地にx座標，C202H番地にy座標をいれ，C000H番地をコールします。

1470行〜1620行は与えられたx，y座標からVRAMのパターン・ジェネレータ・テーブルのアドレスを計算しています。また，これに2000Hをたすことにより，カラーテーブルのアドレスを求めます。

カラーテーブルのデータの上位4ビットはパターンのデータのビットが1の色，下位4ビットはビットが0の色を示しているので，C200H番地のデータとVRAMのカラーのデータの下位4ビットが同じなら，パターンのデータのビットを0にします(1890行)。

違うときは，C200H番地のデータをVRAMのカラーのデータの上位4ビットに埋め込み，パターンのデータのビットを1にします(1740行〜1850行)。

●線を引く

C200H番地にカラー，C201H番地に始点のx座標，C202Hに始点のy座標，C203Hに終点のx座標，C204Hに終点のy座標をいれ，C003H番地をコールします。

直線を引くには，x方向，y方向のうち大きい方の座標は1ずつ変化させ，他の座標は真の値に最も近い整数値を選ぶ単純形DDAというアルゴリズムを使います。

点(x0，y0)から点(x1，y1)へ線を引くことを考えます(**図2-39**参照)。ΔxとΔyを比較するとΔxの方が大きいので，x座標を1度に1つずつ変化させます。y座標の変化は次のようにします。

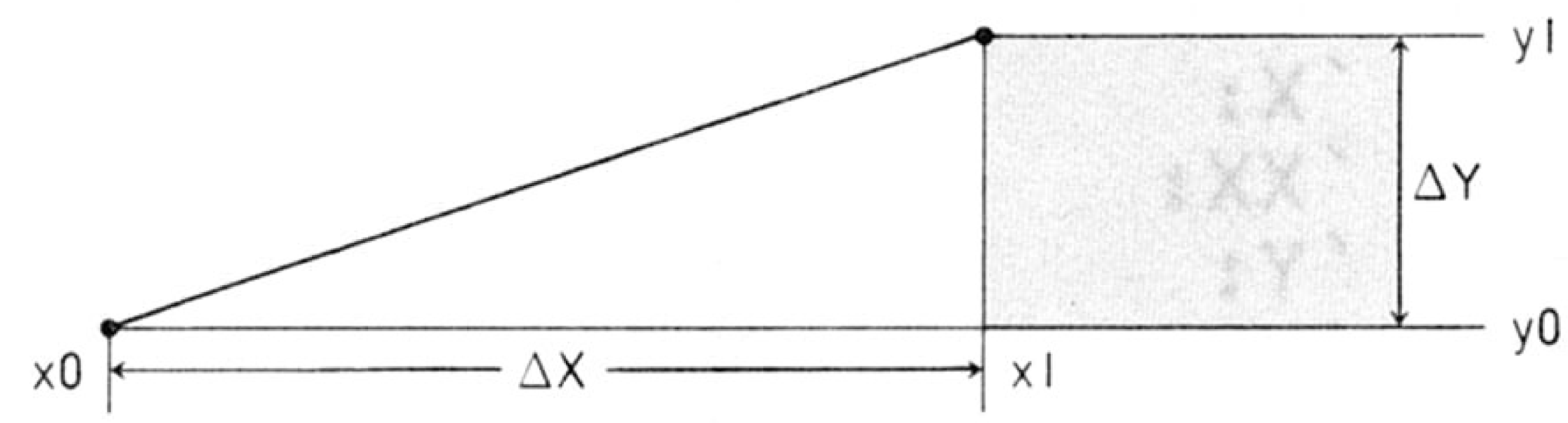
まず，RMという変数を用意し，x座標が1変化するごとにΔyを加えます。これがΔxより大きくなったらy座標を1だけ変化させ，RMからΔxを引きます。

これをx0＝x1になるまで繰り返します。

プログラムの2150行～2290行でΔxとΔyを求め，Δyの方が大きいときはxとyを入れ換えます（2300行～2490行）。

そして，x0＝x1となるまで点を打ちます（2530行～2790行）。

図2-39



●円を描く

C200H番地にカラー，C201H番地に中心のx座標，C202H番地に中心のy座標，C203H番地に始点のx座標，C204H番地に始点のy座標をいれて，C006H番地をコールします。

点(x0，y0)を原点の回りにθだけ回転させた点を(x1，y1)とします。θが微小ならば，

$$x1 = x0 - \theta y0$$
$$y1 = \theta x1 + y0$$

となります。次に同様にして点(x1，y1)をθだけ回転して点（x2，y2）を求め，また次に……というぐあいにして点を打っていくと一周して円になります。

プログラムの2930行～3030行では原点を中心とした座標に円の中心点の座標を足して実際の点の座標を求め，点を打っています。

3040行～3210行で前述の回転の計算をしています。θは1/128としました。1/128をかけるということは，128で割ることなので，3260行からのDIVサブルーチンで，7回右にHLレジスタをシフトしています。

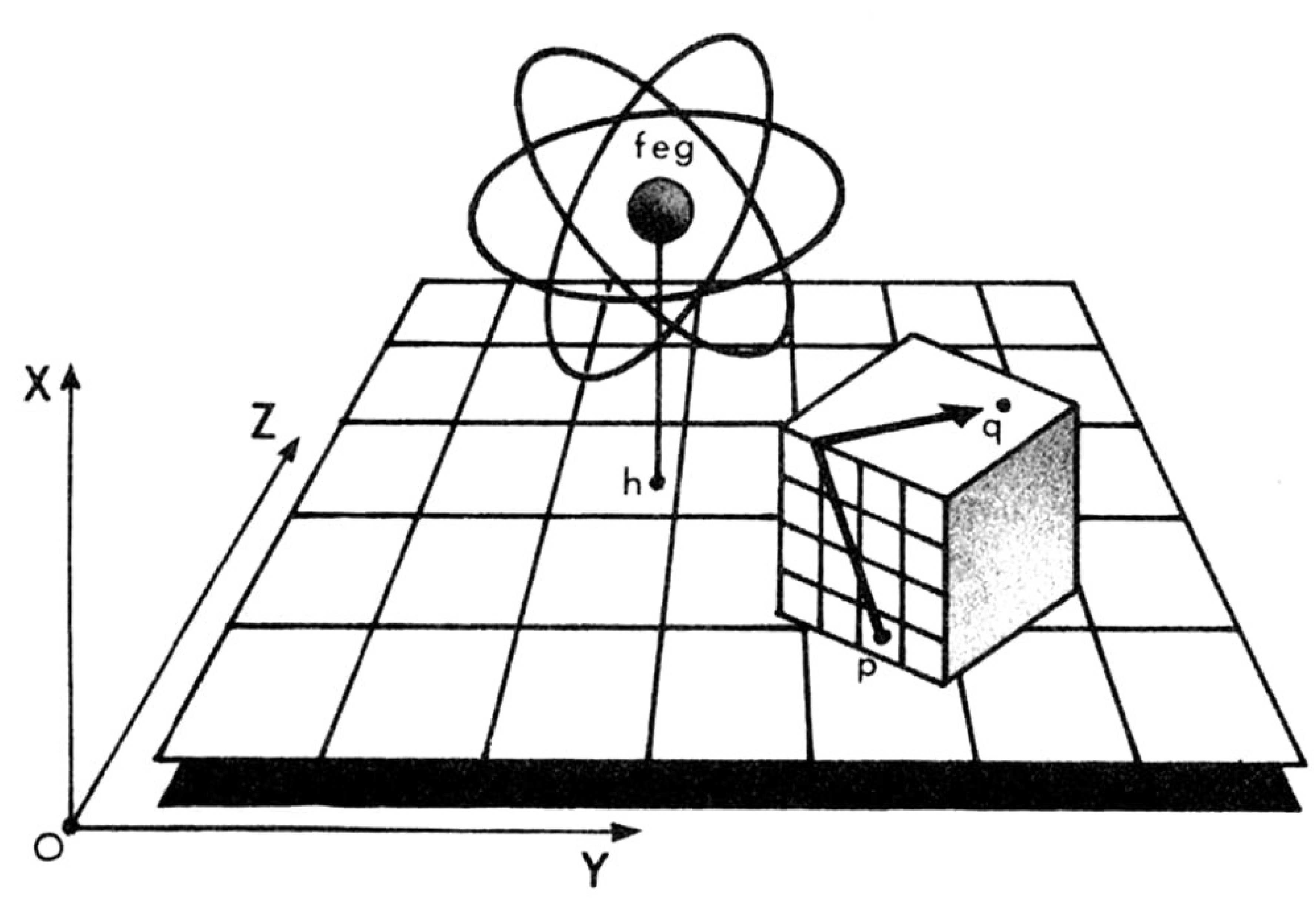
また回転の計算をするときは誤差を少なくするために，x座標の整数部を変数XXに，小数部を変数Xにいれます。y座標も同様に整数部と小数部に分けています。

リスト2-10はテスト・プログラムです。

リスト2-10

```
100 SCREEN 2
110 DEF USR=&HC000
120 DEF USR1=&HC003
```

```
130 DEF USR2=&HC006
140 FOR I=1 TO 100
150   GOSUB 280
160   A=USR(0)
170 NEXT
180 FOR I=1 TO 10
190  GOSUB 280
200  A=USR1(0)
210 NEXT
220 FOR I=1 TO 10
230  GOSUB 280
240  A=USR2(0)
250 NEXT
260 IF INKEY$<>CHR$(13) GOTO 260
270 END
280 POKE &HC200,RND(1)*16
290 POKE &HC201,RND(1)*256
300 POKE &HC202,RND(1)*192
310 POKE &HC203,RND(1)*256
320 POKE &HC204,RND(1)*192
330 RETURN
```



MSXのグラフィック機能を使いこなす

9 中間色の表現について

ＶＤＰでは，透明を含めて全部で16色を画面に表示することができます。一般のパソコンで表示できる色は８色が普通なので，色彩表現の点ではちょっぴり優れているといえるでしょう。

ところが，実際にゲーム・キャラクタのデザインなどをしてみるとわかることですが，16色しか使えないというのは，かなりキビシイ制限なのです。

たとえば人間の皮膚の色を表現しようとすると，ＶＤＰで表示できる16色の中には適当な色がありません。そこで仕方なく，一番それらしい色を選んで代用するということになります。

CGアーティストなどが使う高価なグラフィック・ディスプレイなどには1024色同時表時，あるいはそれ以上といったものがありますが，値段が１ケタから２ケタ以上も違うパソコンではとてもそこまでは望めません。

そこでこういった制限の中で少しでも色表現能力を豊かに見せる方法を紹介しましょう。

1. 中間色表示の原理

まず，その原理について説明します。もし今，あなたの近くにカラーの印刷物があったら，それをじ〜っくりと注意して見てください。あなたが今見ているのもそうですが，一般のカラー印刷物はシアン，マゼンタ，イエロー，そして黒という４色のインキだけで作られています。さらに注意して見ると，それらはそれぞれの色の小さな点の集合でできていることがわかります。

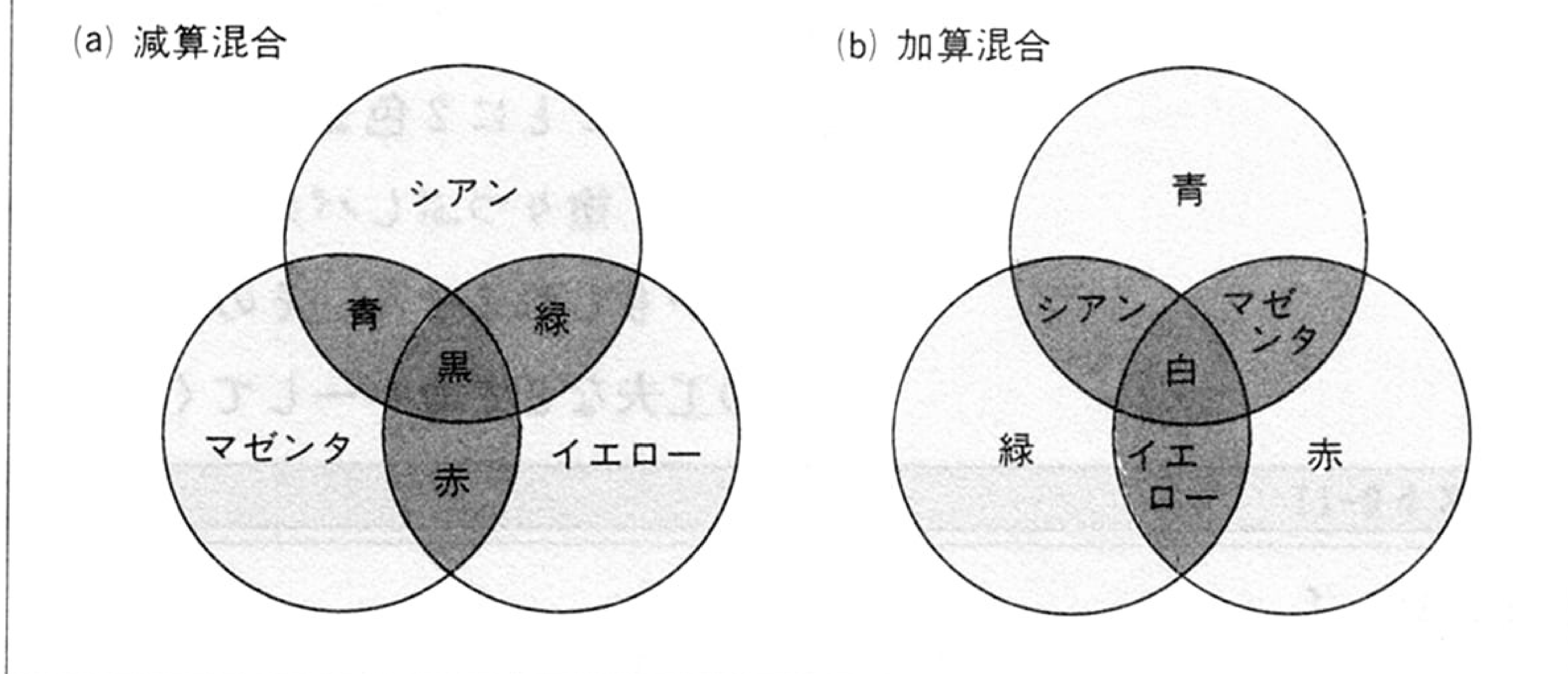
絵の具やインキなどの場合，このシアン，マゼンタ，イエロー(C・M・Y)の３色の比率を変えることですべての色を表現することが可能です。同じようにして，ディスプレイ上の色もレッド，ブルー，グリーンの３色（いわゆるR・G・Bのこと）ですべての色を表現できます。これが小学校の図画工作の時間に習った３原色です。

ところで，絵の具やインクの３原色がC・M・Yなのに対して，ディスプレイ上の色の３原色がR・G・Bなのは，次の理由によります。

絵の具などの場合，色を混ぜ合わせていくと全体の明るさはどんどん暗くなっていきます。これを減算混合といいます。一方でディスプ

レイに表示される色のような光を混ぜ合わせた場合は，全体に明るさを増していきます。これを加算混合といいます(図2-40)。

図2-40　加算混合と減算混合



このように絵の具やインキの場合とディスプレイ上の色の場合は，少しばかり性質が異っていますが，どちらの場合でも３色ですべての色を表現できることに変わりはありません。

ところで先ほど，カラー印刷はC・M・Yのいくつもの点が集まって出来上っているといいました。すなわち実際にインキを混ぜ合わせて色をつくっているわけではありません。こうした細かい点の集まりが実際の人間の目には混じり合って見えているだけなのです。

スーラという画家の『グランドジャットの日曜日』という作品は，点描画法を使ったものとしてあまりにも有名ですが，基本的にはこれとまったく同じ原理によるものです。

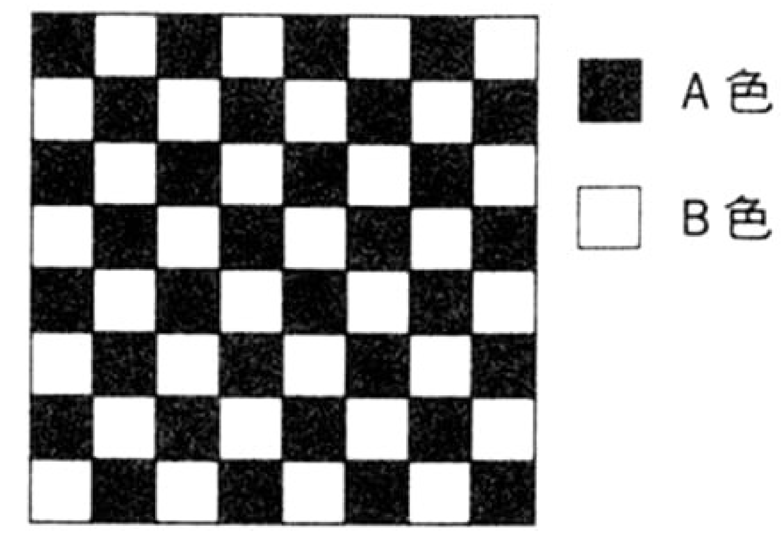
このようにして，ディスプレイ上に小さな間隔で複数色を埋めていけば一見して別の色に見えることになります。

2. 中間色の表示

ここまで説明してきたことをもとにして，実際に中間色表示の実験をしてみることにします。

リスト2-11は画面を**図2-41**のようなパターンで埋めるという，ただそれだけのプログラムですが，注目してもらいたいのはプログラムではなく，画面上に表示されるパターンがどういった色に見えるかということです。

図2-41



16色を２つづつ組み合わせるわけですから，全部で128通りの組み合せができることになります。もっとも実際に表示できる色は128色よりも，もっと少くなりますが，それで

も表現力はかなり拡がるでしょう。**図2-41**のパターンを変えるともっと違った色の見えるはずです。

ところで，こうした方法による中間色表現の欠点は，面の塗りつぶしの場合にしか利用できないということです。またＶＤＰの場合には，横８ドットごとに２色までしか使えない（グラフィックIIモードの場合）ため，塗りつぶしパターンに制約があり，思ったほど自由には使えないかもしれません。その辺はプログラミング・テクニックとデザインの工夫などでカバーしてください。

リスト2-11

```
1000 '
1010 '  PROGRAM COLOR TILE
1020 '  DATE    1-JUL-84
1030 '
1040 COLOR 1,15,15
1050 SCREEN 2
1060 GEN=BASE(12)
1070 COL=BASE(11)
1080 FOR J=0 TO 15
1090  FOR K=0 TO 15
1100   RESTORE
1110   FOR L=0 TO 7
1120    READ P$
1130    VPOKE GEN+J*256+K*8+L,VAL("&H"+P$)
1140    VPOKE COL+J*256+K*8+L,J*16+K
1150 NEXT:NEXT:NEXT
1160 GOTO 1160
1170 DATA AA,55,AA,55
1180 DATA AA,55,AA,55
```

MSXのグラフィック機能を使いこなす

10 グラフィック・エディタ

グラフィックIIモードは，ゲームをつくるという立場からみて一番使い易いモードです。このモードでは，画面上の任意のドットをセットしたり，リセットしたりできるので，キャラクタ・パターン面に絵を描いたりすることができます。

しかし，方眼紙と色鉛筆で絵のパターンをつくるのはかなり大変な作業で，実際の出来上りがどうなるかは，画面に表示してみるまでわかりません。そこで紹介するのがこのグラフィック・エディタです。

このグラフィック・エディタは，グラフィックIIモードのキャラクタ・パターン面に絵を描くことを目的としてつくられたものですが，使い方によってはスプライトのパターンもつくることができます。

基本的に96×64ドットまでの絵を描くことができ，その一部を拡大表示してドットのセット，リセットや色の指定を行いながら絵の編集を進めていきます。

1. グラフィック・エディタの使い方

まず，**リスト2-12**のプログラムを入力します。このプログラムは一部にマシン語サブルーチンを使っているので，RUNをする前に必ずテープにセーブしておいてください。プログラムを実行すると次のように画面に表示して入力を待ちます（以下の入力はすべて英大文字で行ってください）。

リスト2-12

```
1000 '
1010 '   PROGRAM  GRAPHIC EDITOR
1020 '   DATE     1-JUL-84
1030 '
1040 '   WRITTEN by MIPS
1050 '   (c) MIA
1060 '
1070 CLEAR 1000,&HE8FF
1080 DEFINT A-Z
1090 DEFUSR0=&HE900
1100 DEFUSR1=&HE927
1110 '
1120 '   INITIALIZE
1130 '
1140 DIM XX(3),YY(3)
```

```
1150 XX(0)=1:XX(1)=-1:XX(2)=0:XX(3)=0
1160 YY(0)=0:YY(1)=0:YY(2)=-1:YY(3)=1
1170 AD=&HE900
1180 READ A$
1190 IF A$="END" THEN 1230
1200 B=VAL("&H"+A$)
1210 POKE AD,B:AD=AD+1
1220 GOTO 1180
1230 LX=16:LY=80:IA=&HEC00
1240 SCREEN 0
1250 INPUT " ﾃﾞｰﾀ ｦ ﾛｰﾄﾞ ｼﾏｽｶ (Y or N)";A$
1260 IF A$<>"N" AND A$<>"Y" THEN 1250
1270 IF A$<>"Y" THEN 1310
1280 A$="":INPUT "ﾌｧｲﾙ ﾒｲ";A$
1290 BLOAD "CAS:"+A$
1300 Y$="N":GOTO 1330
1310 INPUT" ﾒﾓﾘ ｦ ｸﾘｱｰ ｼﾏｽｶ (Y or N)";Y$
1320 IF Y$<>"Y" AND Y$<>"N" THEN 1230
1330 INPUT" ﾊﾞｯｸ ｶﾗｰ (1-15)";C9
1340 COLOR 15,C9,C9
1350 SCREEN 2,3
1360 GOSUB 2460
1370 IF Y$="Y" THEN A=USR1(0) ELSE A=USR0(0)
1380 A$=CHR$(&HFF):B$=CHR$(&H80)
1390 E$=A$+B$+B$+B$+B$+B$+B$+B$+B$+A$
1400 A$=CHR$(&HC0):B$=CHR$(0)
1410 E$=E$+B$+B$+B$+B$+B$+B$
1420 B$=CHR$(&H40)
1430 B$=B$+B$:B$=B$+B$:B$=B$+B$
1440 E$=E$+A$+B$+A$
1450 SPRITE$(1)=E$
1460 A$=CHR$(&HF0):B$=CHR$(&H90)
1470 E$=A$+B$+B$+A$
1480 SPRITE$(0)=E$
1490 X=0
1500 Y=0
1510 CO=15:IF C9=15 THEN CO=0
1520 IF Y$="N" THEN GOSUB 2560
1530 '
1540 '  MAIN LOOP
1550 '
1560 GOSUB 1680
1570 A$=INKEY$:IF A$="" THEN 1570
1580 K8=ASC(A$)
1590 GOSUB 1780
1600 GOSUB 1840
1610 GOSUB 1970
1620 GOSUB 2060
1630 GOSUB 2170
```

```
1640 GOSUB 2270
1650 IF K8=ASC("!") THEN GOSUB 2820:GOTO 1240
1660 GOTO 1560
1670 '
1680 '  PUT CURSOR (X,Y)
1690 '
1700 X9=X*8+128
1710 Y9=Y*8
1720 IF C9=1 THEN Z=15 ELSE Z=1
1730 PUT SPRITE 1,(X9,Y9),Z,0
1740 RETURN
1750 '
1760 '  PUT LARGE CURSOR
1770 '
1780 IF K8<>ASC("¥") THEN RETURN
1790 IF C9=1 THEN C=15 ELSE C=1
1800 PUT SPRITE 2,(LX-2,LY-3),C,1
1810 FOR I=0 TO 500:NEXT
1820 PUT SPRITE 2,(LX-2,LY-3),0,1
1830 RETURN
1840 '
1850 '  MOVE CURSOR
1860 '
1870 IF K8<28 OR K8>31 THEN GOTO 1960
1880 X9=X+XX(K8-28)
1890 Y9=Y+YY(K8-28)
1900 IF X9<0 THEN X9=0
1910 IF X9>15 THEN X9=15
1920 IF Y9<0 THEN Y9=0
1930 IF Y9>15 THEN Y9=15
1940 X=X9
1950 Y=Y9
1960 RETURN
1970 '
1980 '  PUT THE COLOR
1990 '
2000 IF K8<>32 THEN 2050
2010 PSET (X+LX,Y+LY),CO
2020 X9=X*8+128
2030 Y9=Y*8+1
2040 LINE(X9,Y9)-(X9+6,Y9+6),CO,BF
2050 RETURN
2060 '
2070 '  SET COLOR
2080 '
2090 IF K8<>ASC("C") THEN 2160
2100 A9$=INKEY$:IF A9$="" THEN 2100
2110 K9=ASC(A9$)
2120 IF K9<&H30 THEN 2100
```

```
2130 IF K9>&H39 AND K9<&H41 THEN 2100
2140 IF K9>ASC("F") THEN 2100
2150 CO=VAL("&H"+A9$)
2160 RETURN
2170 '
2180 '  PRESET
2190 '
2200 IF K8<>ASC("D") THEN 2260
2210 IF FRAG =1THEN 2230
2220 PRESET (X+LX,Y+LY)
2230 X9=X*8+128
2240 Y9=Y*8+1
2250 LINE(X9,Y9)-(X9+6,Y9+6),C9,BF
2260 RETURN
2270 '
2280 '  MOVE LARGE CURSOR
2290 '
2300 IF K8<>ASC("@") THEN RETURN
2310 A=USR1(0)
2320 IF C9=1 THEN C=15 ELSE C=1
2330 PUT SPRITE 2,(LX-2,LY-3),C,1
2340 A2$=INKEY$:IF A2$="" THEN 2340
2350 KL=ASC(A2$)
2360 IF KL=13 THEN 2440
2370 A=KL-28
2380 IF A<0 OR A>3 THEN 2340
2390 LX=LX+XX(A)*8
2400 IF LX<16 OR LX>96 THEN LX=LX-XX(A)*8
2410 LY=LY+YY(A)*8
2420 IF LY<80 OR LY>128 THEN LY=LY-YY(A)*8
2430 GOTO 2330
2440 GOSUB 2560
2450 RETURN
2460 '
2470 '  SET THE LINE
2480 '
2490 FOR A=32TO 132 STEP 32
2500 LINE (128,A)-(255,A),15
2510 NEXT A
2520 FOR A=127 TO 256 STEP 32
2530 LINE (A,4)-(A,128)
2540 NEXT A
2550 RETURN
2560 '
2570 '  RE-DRAW  ROUTINE
2580 '
2590 AD=(LY-80)*12+LX-16
2600 FOR X7=0 TO 1
```

```
2610 FOR Y7=0 TO 1
2620 FOR Y6=0 TO 7
2630 Y8=AD+X7*8+Y6+Y7*96
2640 B=PEEK(Y8+&HED60)
2650 C=INT(B/16)
2660 E=B MOD 16
2670 A$=RIGHT$("0000000"+BIN$(PEEK(Y8+&HEA60)),8)
2680 FOR D=0 TO 7
2690 D$=MID$(A$,D+1,1)
2700 CO=C:IF D$="0" THEN CO=E
2710 X=D+X7*8
2720 Y=Y6+Y7*8
2730 GOSUB 2020
2740 GOSUB 1700
2750 NEXT
2760 NEXT
2770 NEXT
2780 NEXT
2790 X=0:Y=0
2800 PUT SPRITE 2,(LX-2,LY-3),0,1
2810 RETURN
2820 '
2830 '  BSAVE DATA
2840 '
2850 A=USR1(0)
2860 A$=""
2870 INPUT "ﾌｧｲﾙ ﾒｲ";A$
2880 IF A$="" THEN GOTO 2920
2890 PRINT "ﾃｰﾌﾟ ｦ ｾｯﾄ ｼﾃ､RETURN KEY ｦ ｵｼﾃｸﾀﾞｻｲ"
2900 IF INKEY$<>CHR$(13) THEN 2900
2910 BSAVE"CAS:"+A$,&HEA60,&HF05F
2920 RETURN
2930 '
2940 '  MACHINE CODE DATA
2950 '
2960 DATA 21,60,EA,11,10,0A,CD,13
2970 DATA E9,21,60,ED,11,10,2A,CD
2980 DATA 13,E9,C9,06,08,C5,D5,E5
2990 DATA 01,60,00,09,E3,CD,5C,00
3000 DATA E1,D1,C1,14,10,EF,C9,21
3010 DATA 10,0A,11,60,EA,CD,3A,E9
3020 DATA 21,10,2A,11,60,ED,CD,3A
3030 DATA E9,C9,06,08,C5,E5,D5,01
3040 DATA 60,00,EB,09,E3,EB,CD,59
3050 DATA 00,D1,E1,C1,24,10,ED,C9
3060 DATA END
```

①新しく絵を描く場合

データ　ヲ　ロード　シマスカ（Y or N）？ N RETURN
メモリ　ヲ　クリアー　シマスカ（Y or N）？ Y RETURN

②以前に描いた絵を編集する場合

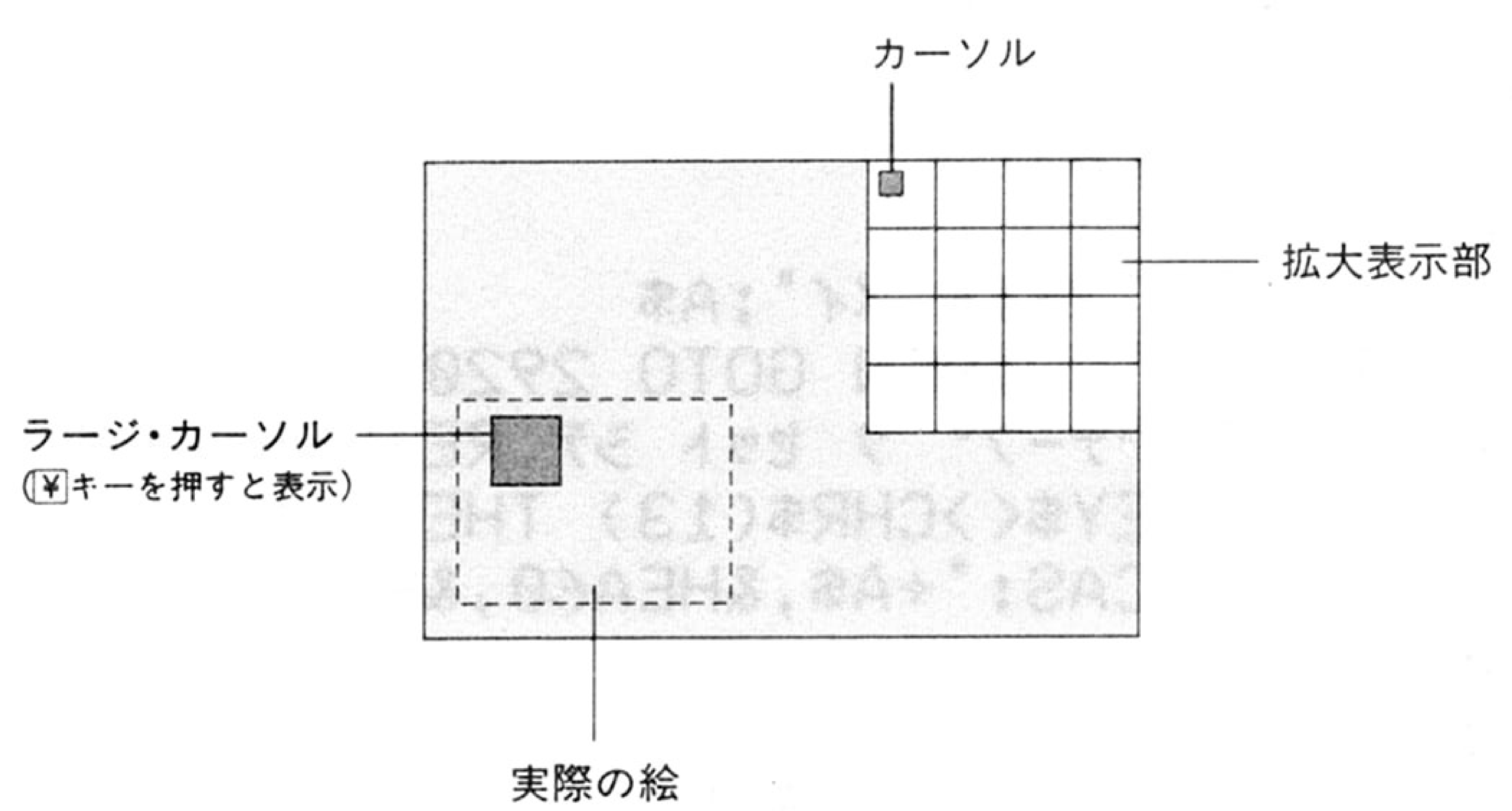
データ　ヲ　ロード　シマスカ（Y or N）？ Y RETURN
ファイル　メイ　？○○○ RETURN
（○○○は，セーブしたデータのファイル名）

次に，

バック　カラー（1－15）？

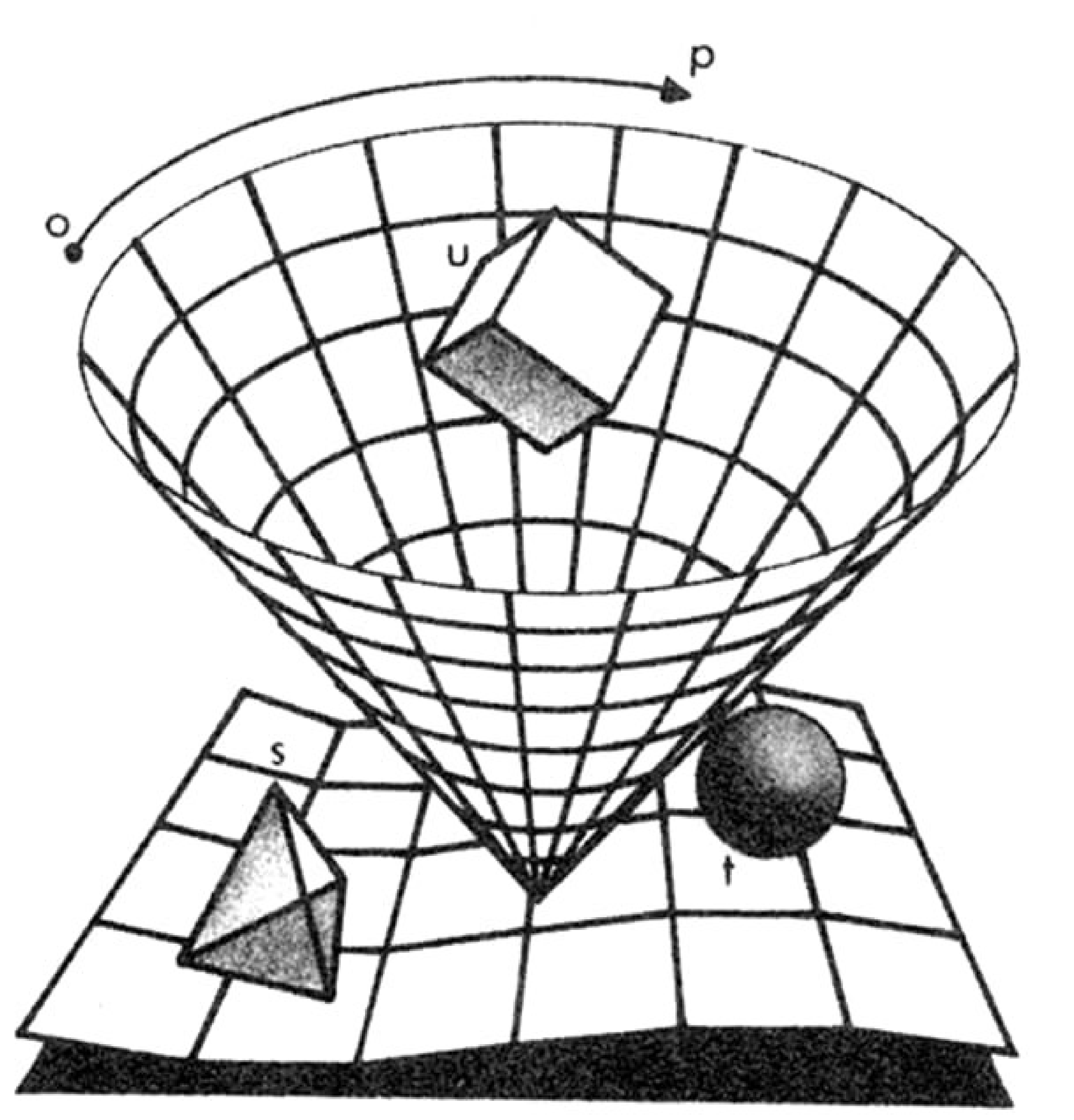
と聞いてくるので1～15の数字を入力し，バック・カラーの指定を行います。これが終わると画面は**図2-42**のようになり，以下の各コマンドで絵の編集ができるようになります。

図2-42　エディティング中の画面



キー	内　容
↑ ↓ ← →	カーソルを8方向に移動する。
SPACE	カーソルの位置のドットをセットする。
D	カーソルの位置のドットをリセットする。
C	Cを押したあと，続けて0～9，A～Fを入力することにより，カラーの指定を行う。

	以後セットされるドットの色はこのカラーになる。０～９，Ａ～ＦはＶＤＰのカラーコードと同じである。
¥	押している間，ラージ・カーソルを表示する。ラージ・カーソルは現在どこを拡大表示しているかを示している。
@	ラージ・カーソルを移動し，その部分を拡大表示する。@を押してから，カーソル・キーを用いて，拡大表示したい部分へラージ・カーソルを持っていき，RETURN キーを押せばその部分が編集できるようになる。
!	エディット作業を終了する。できあがった絵のデータをセーブするために，レコーダーにテープをセットし，録音状態にする。ファイル名を入力して RETURN キーを押すと，セーブが開始される。



2. 使用上の注意

ＶＤＰのグラフィックIIモードでは，横８ドットごとに２色までの色しか指定することができません。このエディタを使って３色目を指定すると絵の一部が乱れ，拡大表示部と実際の画面が一致しなくなります。その場合は，@キーを押し，続けて RETURN キーを押してください。これで拡大表示部と画面が一致するようになります。８ドットごとに２色しか使えないというこの制限は，絵を描く場合，かなりきびしいのですが，絵の方のデザインを工夫するなりして解決してください。

どうしても３色以上使いたい場合は，そのためにスプライトを１枚分定義して，画面に重ね合わせるなどといった方法で解決することもできます。

3章 MSXのサウンド機能を使いこなす

PSG(AY-3-8910)の使いかた

ゲームに音は大切な要素です。この章ではMSXのサウンド機能の中核をなすPSGを取り上げ，その機能とゲームへの応用を考えてみることにしましょう。

MSXのサウンド機能を使いこなす

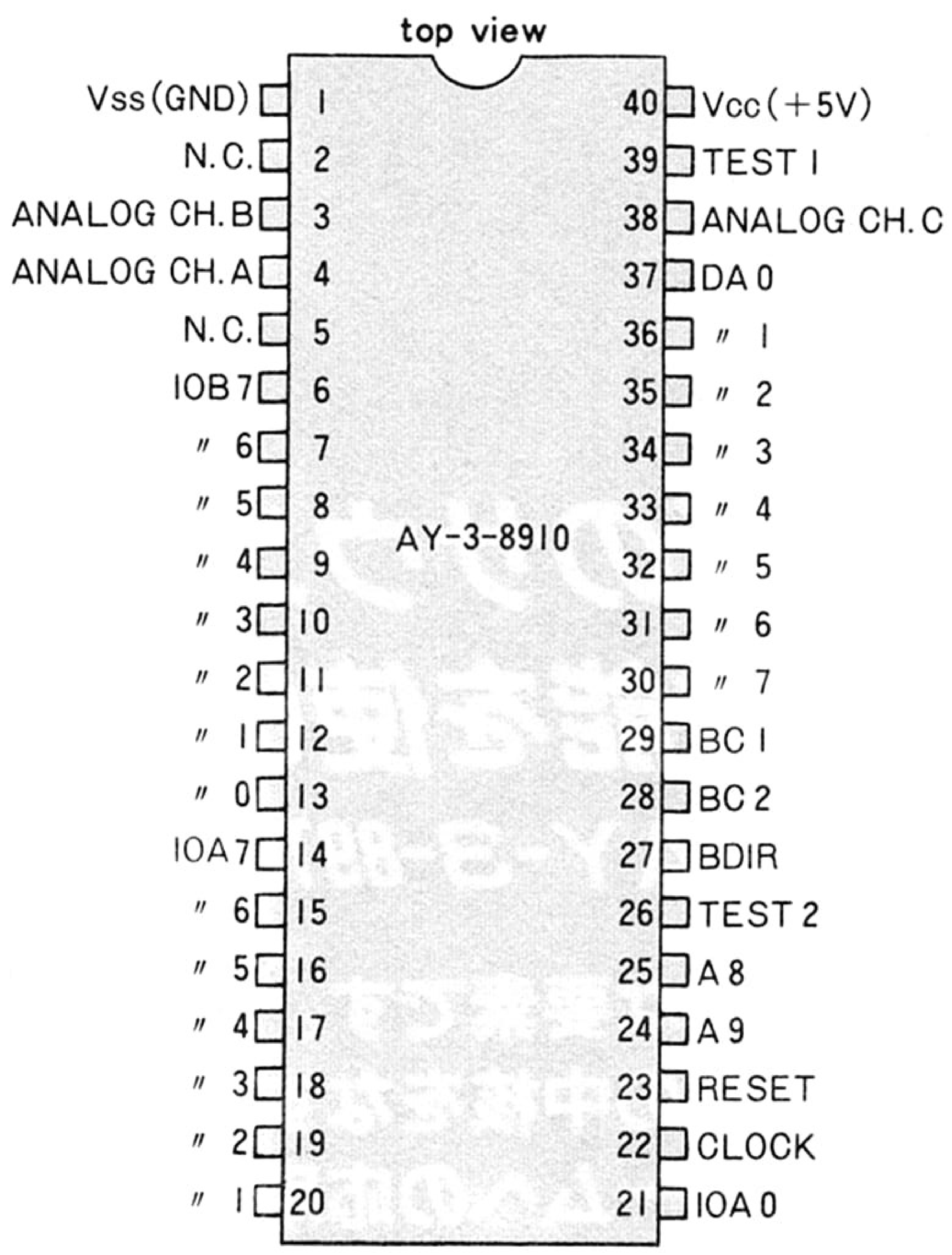
1 PSGについて

ＰＳＧとはProgrammable Sound Generatorの頭文字を取ったもので，早い話が皆さんよくご存知のシンセサイザをＬＳＩ化(図3-1)したものなのです。そんなことから別名ワンチップ・シンセサイザとも呼ばれ，手軽に使えることからゲーム・センターにあるゲーム機には必ずといってよい程これが組み込まれています。

シンセサイザという名前のとおり音楽の演奏なんかは得意中の得意といってよく，ドカーンとかバキューンとかいった派手な効果音なども，工夫次第でいろいろと作り出すことが可能です。こうした効果音の発生などはゲームにはもってこいの機能といったところでしょう。

ところでここでいうＰＳＧとは，ＭＳＸの標準品であるＧＩ社のＡＹ－３－8910のことをいいます。ＰＳＧの場合はＶＤＰと違い，同等の製品が数多く出回っているので混同しないようにしてください。以下本書でＰＳＧと言った場合には，それはAY-3-8910を指します。

図3-1　ＰＳＧピン配置図



Vss	GND
Vcc	5 V
N.C.	未使用
CH.A	チャンネルＡ音声出力
CH.B	チャンネルＢ音声出力
CH.C	チャンネルＣ音声出力
IOA 0 ～IOA 7	ポートＡ入出力
IOB 0 ～IOB 7	ポートＢ入出力
DA 0 ～DA 7	データ・バス
BC 1,BC 2	バス・コントロール
BDIR	バスの方向
CLOCK	クロック入力(MSXの場合 1.7897725MHz)
A 8，A 9	チップ・セレクト
TEST 1,TEST 2	メーカーのテスト端子，接続しない
RESET	リセット

2 MSXのサウンド機能を使いこなす

音に関する基礎知識

ＰＳＧの使い方をマスターしたとしても，それだけで自分のイメージに合う音作りができるわけではありません。音作りをする上では，音の性質や特徴を理解しておく必要があります。まずは，音について説明していきましょう。

音の高さ（音程），音色（おんしょく），音の大きさ（音量）は音の３要素と呼ばれます。このうち音量は振幅の大きさで決まります。

1.音程

私達が一般に音楽で扱う音には，それぞれ名前があって，ド・レ・ミ…とかＣ・Ｄ・Ｅ…といった呼び方（これを音名という）をします。こうした音楽で用いられる音を全部ひっくるめて楽音と呼び，五線紙のオタマジャクシで表わすことはご存知のとおりです。

この楽音と周決数の間には，一定の取り決めがあって，Ａの音（ピアノの鍵盤上で左から５番目のラの音）を440Hzとすることが定められています。これは，時報のピッピッピッという音の高さと同じで，ちなみに最後のポーンという音の周波数は880Hz（１オクターブ高いラの音）です。

普段，私達が使っている音階は，半音高くなると周波数が$\sqrt[12]{2}$倍になるように構成されています。１オクターブは12の半音を含んでいるので，１オクターブ高い音はちょうど周波数が２倍になり，逆に１オクターブ低い音は½になります。

というわけで，音の高さは周波数によって決まるわけです。

2.音色

私達の耳は，ピアノとバイオリンの音の違いを聞きわけることができます。これはピアノの出す音の波形とバイオリンの波形が異なっているからです。波形は倍音の含み方によって決定されます。したがって，音色の違い＝波形の違い＝倍音の含み方の違いといえます。

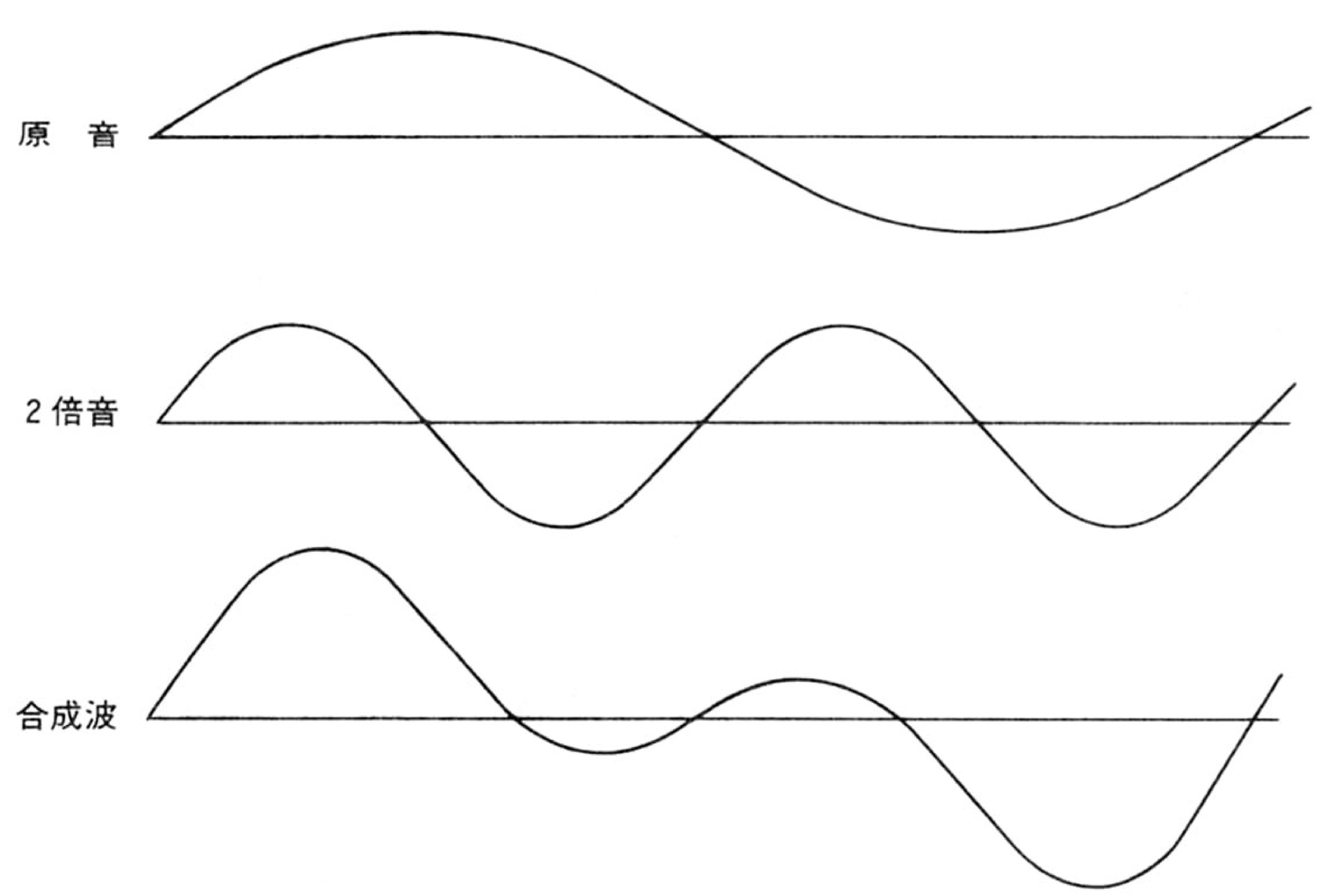
さきほど，音程は音の周波数によって決まると言いましたが，この音程を決める周波数の音を原音（基音とか純音とも呼ばれる）と呼び，波形はサイン・カーブ（正弦波形）になっています。一般に，音はこの原音の他に様々な周波数の音を含みます。これらは，原音のｎ倍の周波数を持ち，倍音と呼ばれています。**図3－2**は原音と２倍音を合成

したものです。もちろん，

音の高さ（音程）＝原音の周波数

なので，倍音をいくら含んでも音程は変わりません。こうして，合成して得られた波形が実際私達の耳に届くわけですが，この合成波が倍音の含み方によって波形が変わることがわかるでしょう。

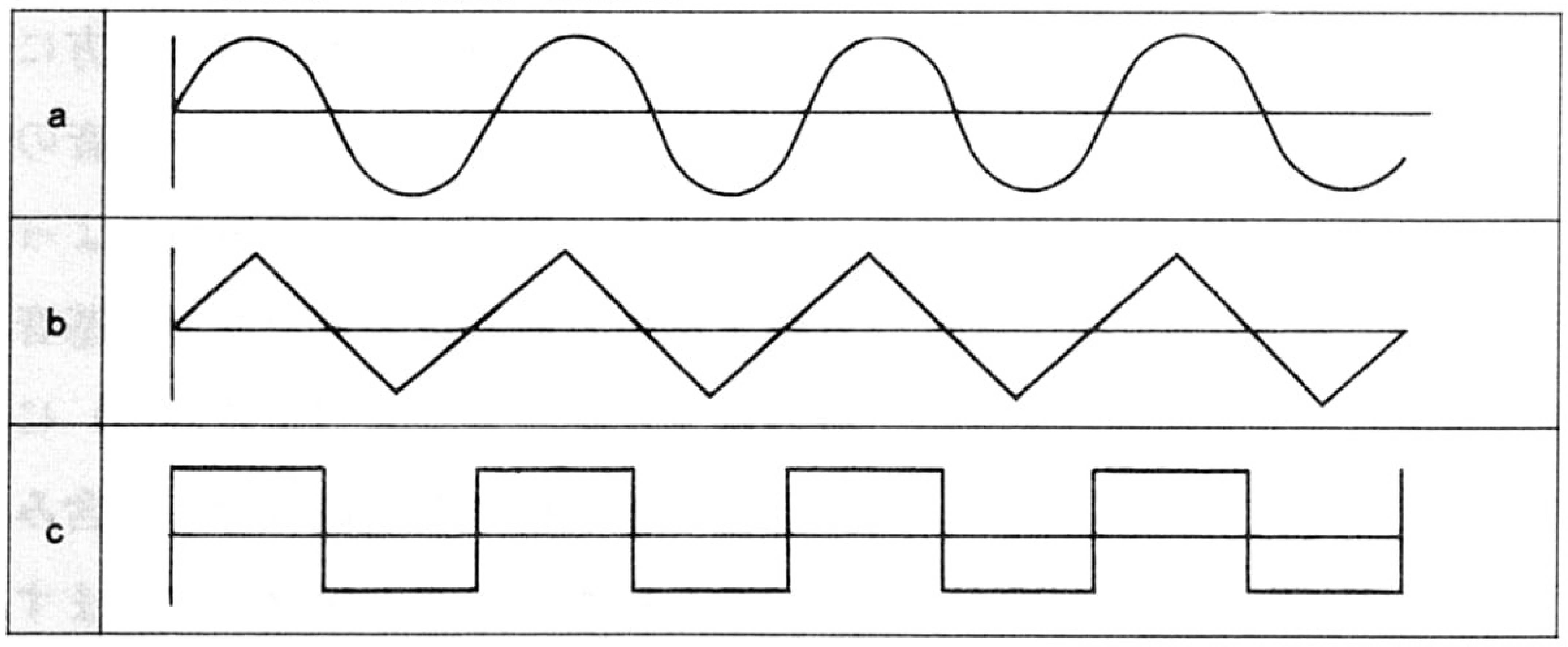
図3-2　波形の合成



さて，シンセサイザで音の合成によく使われる基本的な波形について述べましょう。**図3-3**(a)の波は前述した正弦波で，他の周波数成分をいっさい含みません。音叉を叩いたときに出る音はこの原音です。

図3-3の（b）と（c）は，それぞれ三角波，短形波などと呼ばれますが，これらも原音に多くの倍音を合成して作られています。特に矩形波は，三角波よりも高周波成分が多く，原音と比べると多少ぎらついた音に聞こえます。ＰＳＧから出力されるのはこの矩形波です。

図3-3　正弦波，三角波，短形波



3.エンベロープ

音の３要素について述べてきましたが，これだけで音の感じを表わすことはできません。ある音を出したときに，これが消えるまでの音量の変化をエンベロープと呼びます。音叉を叩いたときのエンベロープを**図3－4**に示します。このエンベロープは楽器によって異なり，これをエンベロープ・パターンまたはエンベロープ・カーブと呼んでいます。

エンベロープ・パターンは，４つに分割されそれぞれアタック（立ち上がり），ディケイ（減衰），サスティン（持続），リリース（消滅）と呼ばれます（**図3-5**）。これはそれぞれの英語の頭文字をとってＡＤＳＲと総称しています。

図3-4　音叉のエンベロープ・パターン

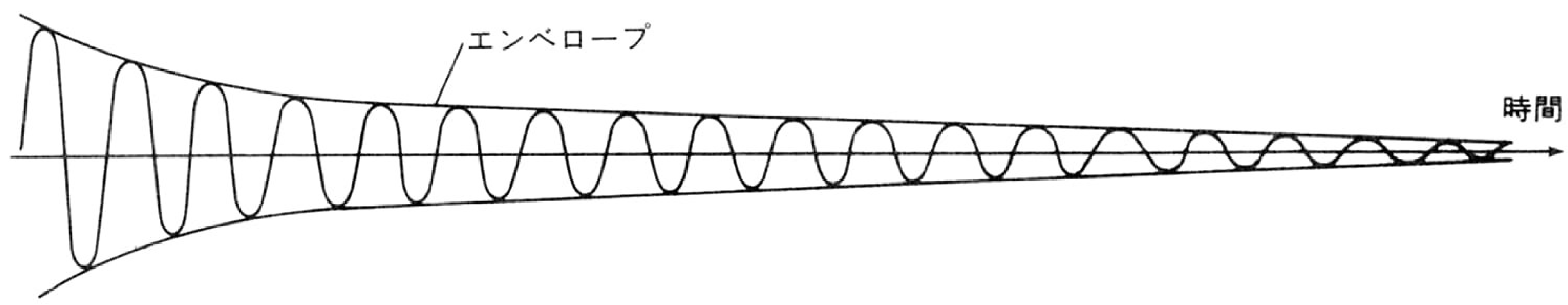
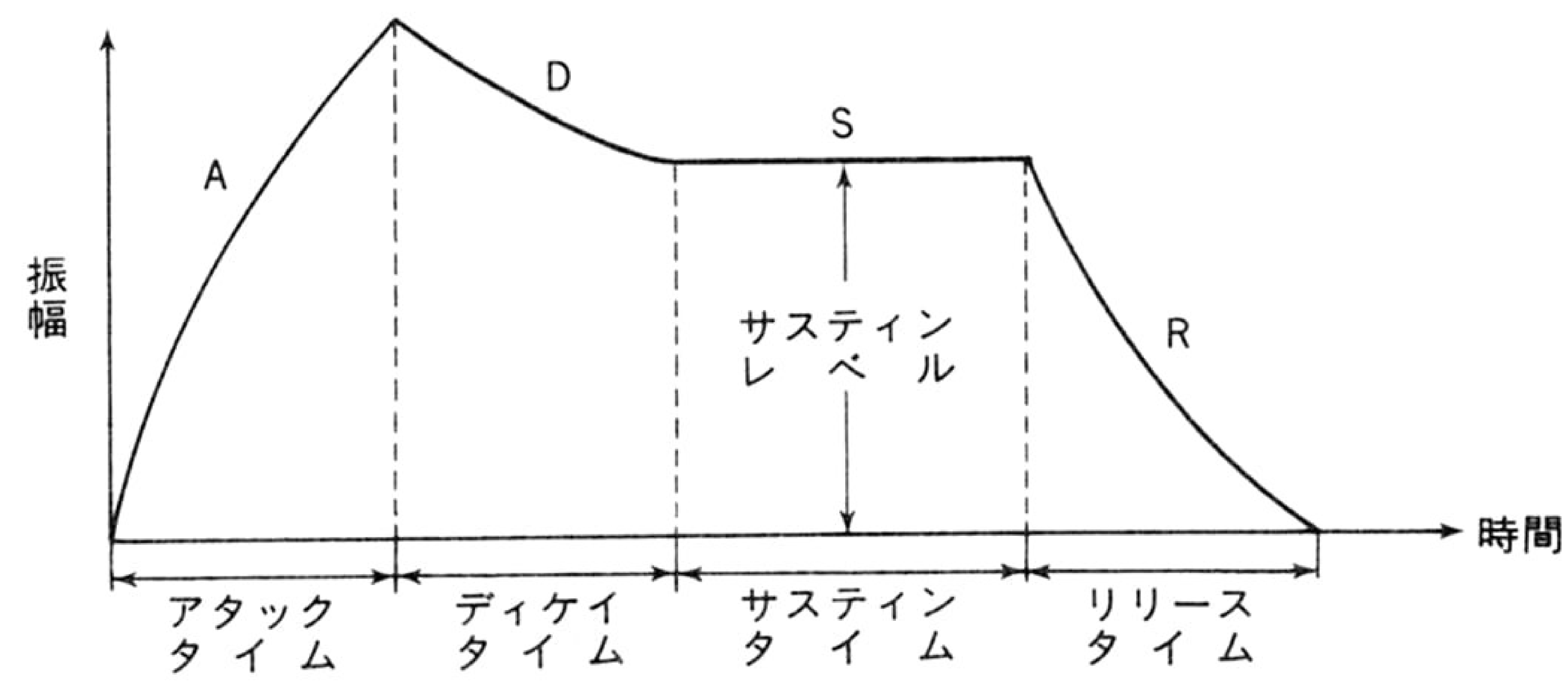
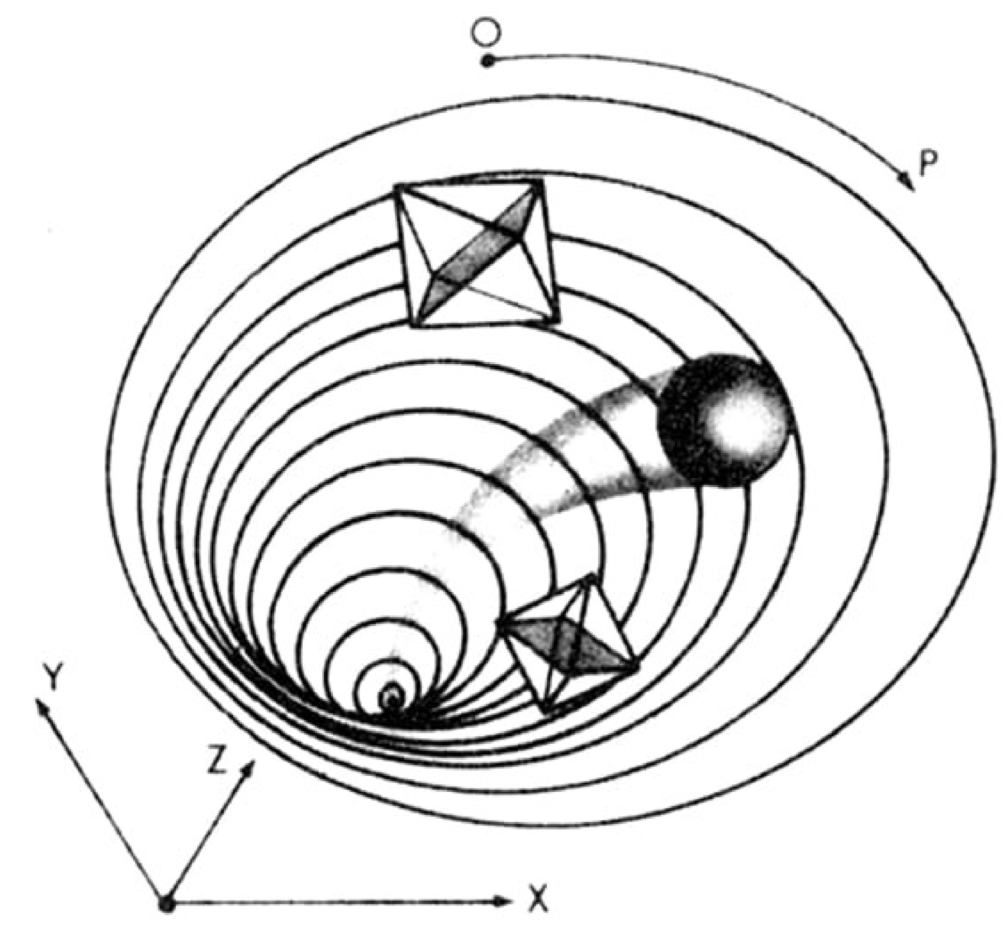


図3-5　ADSR

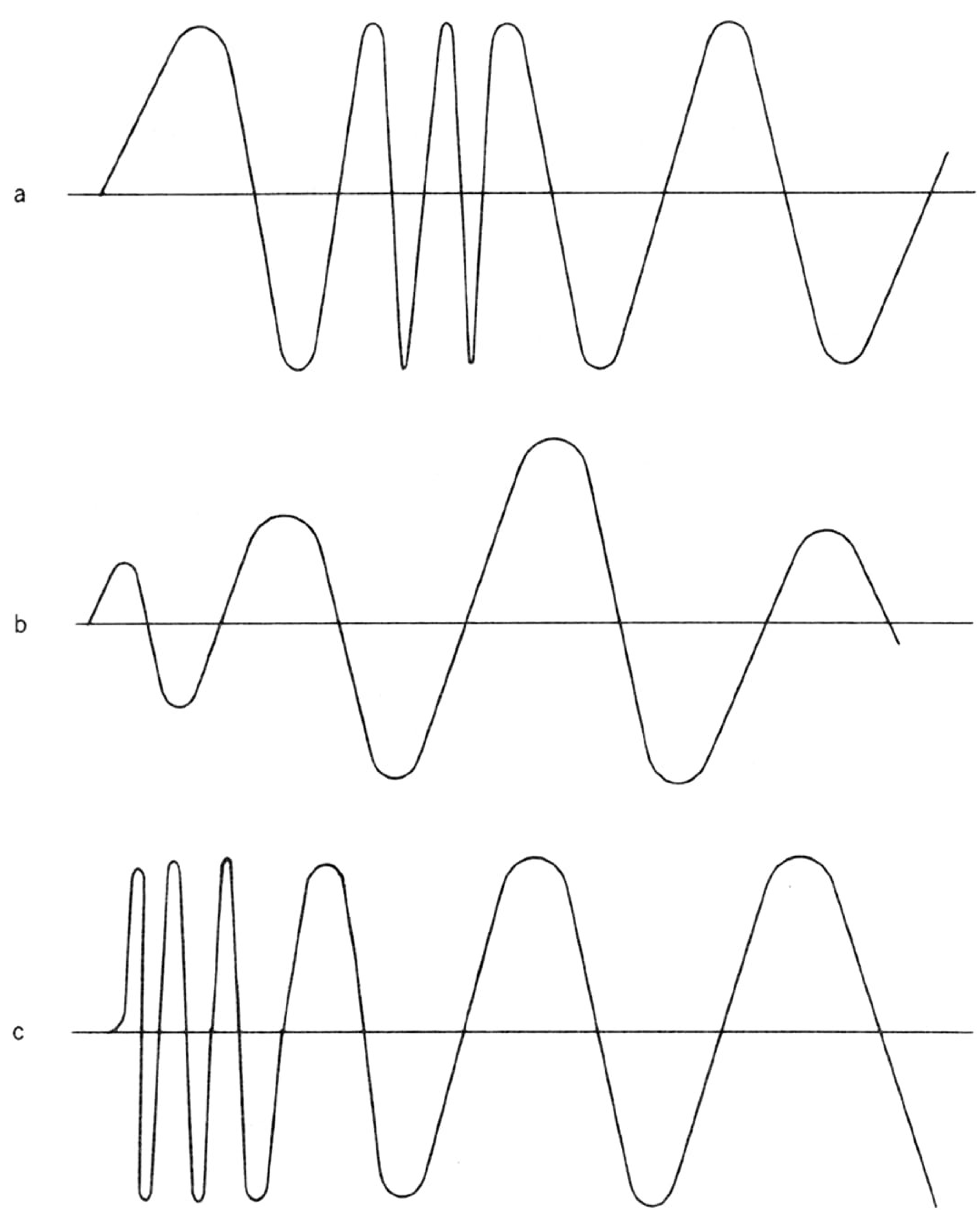




4.効果音

ビブラート，トレモロ，ポルタメントといった効果音を**図3-6**に示します。トレモロは，強くかけた場合（振幅を深くかける）マンドリンの音の大きさの変化に似ているのでマンドリン効果とも呼ばれます。

図3-6　ビブラート，トレモロ，ポルタメント



5.ノイズ

特定の周波数を持たない音がノイズで，逆に言うと数多くの周波数を含んでいる音といえます。こうしたノイズのなかで，一般によく知られているのがホワイト・ノイズとピンク・ノイズです。

ホワイト・ノイズはすべての帯域で同じエネルギーを持ち，『シャー』といった感じに聞こえます。ＦＭ放送の局間ノイズは，このホワイト・ノイズの代表といえるでしょう。一方のピンク・ノイズは各帯域で振幅の同じ音を集めたもので，風の音のような『ゴー』といった感じの音です。

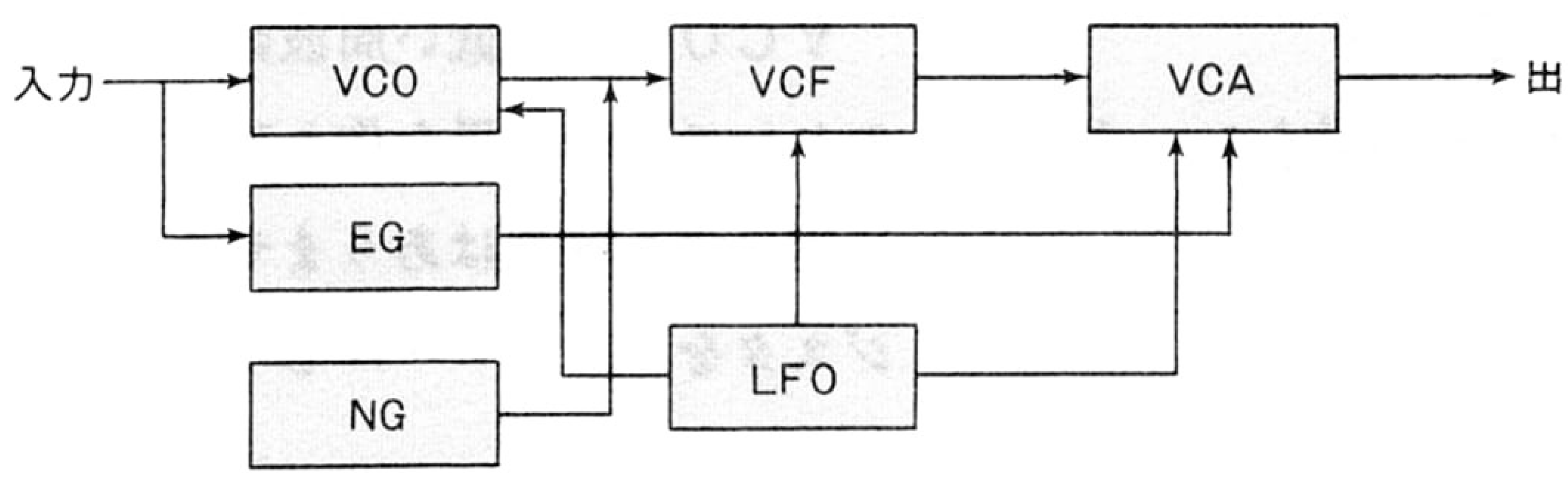
MSXのサウンド機能を使いこなす

3 シンセサイザの基礎

ワンチップ・シンセサイザという名前からも解かるように，ＰＳＧを使いこなすには，やはりシンセサイザについての知識が必要です。最近ではデジタル・タイプのシンセサイザが普及してきて，回路も基本構成もかなり変化しているようですが，ＰＳＧの構成は今までのアナログ・タイプのシンセサイザとほとんど同じです。そこでまず基本的なシンセサイザの機能を解説しておくことにしましょう。

図3-7はシンセサイザの基本構成を示したものです。

図3-7　シンセサイザの基本構成



1.VCO（電圧制御形オッシレータ）

電圧を変化させて任意の周波数を発振させ，ＶＣＦやＶＣＡに音の素材を提供するものです。

一般に楽器として売られているシンセサイザは，ＶＣＯの部分で発振させる波形を選択することができます。こうした波形は音色と密接な関係を持っていて，同じ高さの「ド」の音であっても，違った楽器で演奏すると，波形の違いからそれぞれの音色を聞きわけることができるのです。

2.VCF（電圧制御形フィルタ）

ＶＣＯやＮＧ（ノイズ・ジェネレータ）で作られた音の素材の中から，一定の周波数だけをカットしたり，逆に一部を取り出したりします。オーディオ・アンプのトーン・コントロールに相当する部分だと思っていただければいいでしょう。

ＶＣＯで作られた波形を整形して音色を作り出す重要な部分ですが，残念ながらＰＳＧにはこのＶＣＦに相当する機能がありません。そんなわけでＰＳＧを使って，バイオリンやトランペットのような音を作

り出すということはできませんが，ゲームのサウンド出力に使うのがここでの目的なので，この際は我慢することにしましょう。

3.VCA
（電圧制御形アンプ）

音声の大きさをコントロールします。もっともただ単に音量を調節するというのではなく，ＥＧやＬＦＯで作られた信号を利用することで，いろいろな効果を作り出すことができます。

4.NG
（ノイズ・ジェネレータ）

特定の周波数を持たない音がノイズで，このノイズを発生させるのがＮＧです。ノイズは爆発音などの効果音を発生させる際に利用されます。

5.LFO
（低周波オシレータ）

ＶＣＯよりも低い周波数を発振する回路です。一般にはビブラートやトレモロの効果を作り出すときに使用されます。ＰＳＧにはＬＦＯに相当する機能はありませんが，ＬＦＯの代わりにソフトウェアでレジスタをコントロールしてやることで同様な効果を実現できます。

6.EG
（エンベロープ・ジェネレータ）

ＶＣＡをコントロールして，音声の大きさを一定のパターンで変化させます。もっともこれだけではいったいどんな目的に使うのか解からないと思うので，例を上げて説明しましょう。

たとえばピアノの場合,キーを押した瞬間に音は鋭く立ち上がり（アタック），ゆっくり減衰(ディケイ)しますが，フルートの場合は立ち上がりは比較的ゆっくりで，息を吹き込んでいる間はある一定のレベルを保持します。

こうした楽器や音の持つ特定のパターンを作り出すのがＥＧです。つまり，前に説明したＡＤＳＲを設定するものです。

4 MSXのサウンド機能を使いこなす

PSGの基本機能

ＰＳＧはシンセサイザとしての，必要最低限かつ十分な機能を備えています。ここでその特徴をまとめてみましょう。

①３つのトーン・ジェネレータを持ち，３声までの同時出力が可能である。

②ノイズ・ジェネレータを持つ。そのため効果音の発生が容易である。

③これらトーン・ジェネレータおよびノイズ・ジェネレータの音量を個別にソフトウェアでコントロールできる。

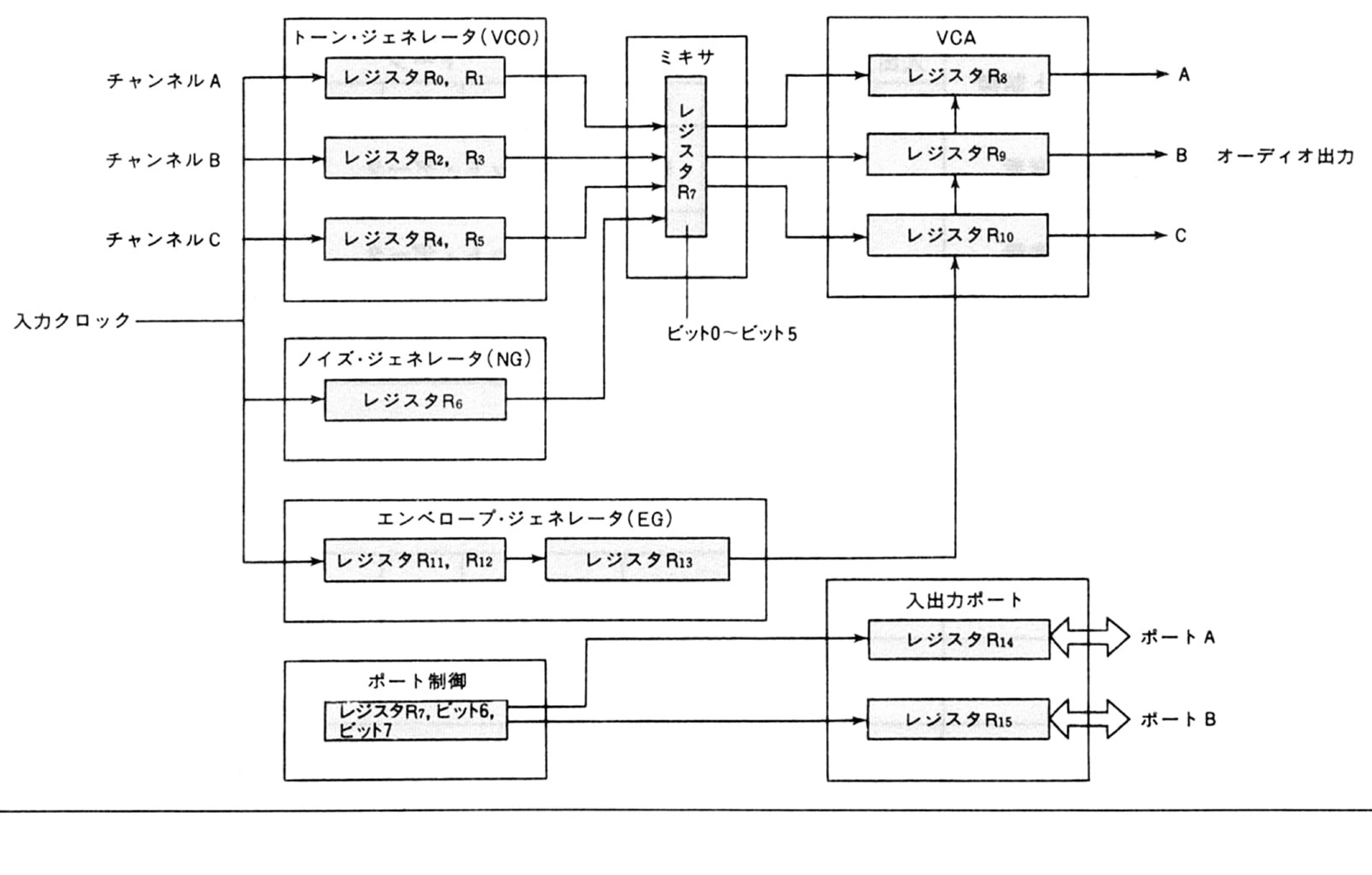
④エンベロープ・ジェネレータを持ち，それによるエンベロープ・パターンの指定が可能。

⑤３つのトーン・ジェネレータ，ノイズ・ジェネレータの出力をミキシングする機能がある。

⑥２つの８ビット・パラレル入出力ポートを持つ（ＭＳＸではこれら入出力ポートをジョイスティックの入力，カセットの入力などに利用している）。

ＰＳＧは全部で16個の８ビットレジスタを持ち，上記の機能はすべて，これらレジスタに書き込まれる値によってコントロールされます。

図3-8　PSGブロック・ダイアグラム



MSXのサウンド機能を使いこなす

5 PSGのレジスタ

ムーグやアープといった，初期の頃のシンセサイザ（富田勲といっしょに写真にうつっているようなヤツ）を見ると，つまみやスイッチがやたらたくさん付いているのが目につきます。もちろんこういったスイッチ類をコントロールして音を合成するわけですが，ＰＳＧではこうしたつまみやスイッチに相当するのが，16個のレジスタ群（そのうちサウンド機能に関係するのは14個）です(表3-1)。

表3-1　ＰＳＧのレジスタ一覧

レジスタ	機　能	7	6	5	4	3	2	1	0	
R_0	チャンネルＡの周波数	下位８ビット・データ								
R_1						上位４ビット・データ				
R_2	チャンネルＢの周波数	下位８ビット・データ								
R_3						上位４ビット・データ				
R_4	チャンネルＣの周波数	下位８ビット・データ								
R_5						上位４ビット・データ				
R_6	ノイズの平均周波数				５ビット・データ					
R_7	ミキシング，ポート制御	入出力選択		ノイズ			トーン			対応するビットが１のときオフ，０のときオン
		ポートＢ	ポートＡ	Ｃ	Ｂ	Ａ	Ｃ	Ｂ	Ａ	
R_8	チャンネルＡの音量				Ｍ	４ビット・データ				Ｍ＝０のとき下位４ビット・データによる音量調節 Ｍ＝１のとき エンベロープ作動
R_9	チャンネルＢの音量				Ｍ	４ビット・データ				
R_{10}	チャンネルＣの音量				Ｍ	４ビット・データ				
R_{11}	エンベロープ周期	下位８ビット・データ								
R_{12}		上位８ビット・データ								
R_{13}	エンベロープ波形					CONT	ATT	ALT	HOLD	
R_{14}	ポートＡ									R_7の入出力選択が１のとき出力，０のとき入力
R_{15}	ポートＢ									

1.レジスタR_0〜R_5

ＰＳＧのレジスタのうち，R_0からR_5まではトーン・ジェネレータ(ＰＳＧではＶＣＯのことをこう呼ぶ）で発振される周波数をコントロールするのに使われています。ＰＳＧは３つのトーン・ジェネレータを持ち，それぞれをチャンネルＡ，Ｂ，Ｃと呼びます。

レジスタは各チャンネルに対しR_0とR_1，R_2とR_3，R_4とR_5をそれぞれペアにして使用し，そこに下位８ビット，上位４ビットの計12ビットのデータを書き込みます。チャンネルＡを例にとると，R_0に書き込まれた８ビット・データと，R_1のビット０から３での間に書き込まれた４ビットのデータの合計12ビットのデータが発振周波数を決定します。それ以外のR_1のビット４から７までの４ビット分については，たとえ書き込まれても無視されます。

さて発振される周波数(音の高さ）ｆとレジスタに書きこんだ12ビットのデータＤとの関係は次の式で表わすことができます。

$$f = \frac{fclock}{16 \times D}$$

fclock：PSGへのクロック入力（1.7897725MHz）

f　　：出力される周波数

D　　：書き込まれる12ビットのデータの値

MSXの場合，fclockの値は1.7897725MHzということになっているので，実際に出せる最高の音はデータの値が１のとき111.8KHzで，最低の音は4095のときの27.3Hzです。これは可聴帯域の音の全てをカバーしています。

2.レジスタR_6

レジスタR_6はノイズの平均周波数をコントロールするものです。R_6に書き込まれたデータは，下位５ビットのみが有効で，上位３ビットについては無視されます。したがってR_6を使用する場合に，実際に有効となる値は１から31までということになります。ここでノイズの平均周波数をf，R_6に書き込まれるデータをＤとすると，

$$f = \frac{fclock}{16 \times D}$$

fclock：PSGへのクロック入力（1.7897725MHz）

f	：出力されるノイズの平均周波数
D	：R_6の下位5ビットの値

といった関係が成り立ちます。実際に出力されるノイズの平均周波数は，書き込むデータが1のときに111.9KHz，31のときに3.6KHzで，データが小さいときはホワイト・ノイズの様に聞こえ，逆に大きくなるとピンク・ノイズに近くなります。

3.レジスタR_7

トーンとノイズのミキシングをする他，ＰＳＧの持つ２つのポートの入出力の方向を決定します。

このレジスタR_7のビット０から２までがトーンの出力を決定するスイッチで，ビット３から５までが同様にノイズの出力を決めるスイッチです。それぞれのビットに０を書き込むとONになり，1を書き込むとOFFになります。たとえばチャンネルＢをトーンとノイズのミックス出力にしたい場合は，ビット４と１にそれぞれ０を書きこめばよいわけです。

さてまだ説明していないビット６とビット７の使い方ですが，これは入出力ポートの方向を決定するのに使用されています。ＭＳＸではポートＡを入力，ポートＢを出力として使っているので，単にビット６＝０，ビット７＝１にしておかなければなりません。ですから，R_7にデータを書き込むときはビット６と７を変化させないように注意してください。

4.レジスタR_8～R_{10}

A,B,Cの各チャンネルの音量をコントロールします。音量は４ビットで表わし，15が最大で０が最少(無音)です。ただし，各レジスタのビット４を１にすると音量の調節はエンベロープ・ジェネレータに任されることになり，その場合は下位４ビットに設定した音量は無視されます。したがってエンベロープを使用しないときは，ビット４は単に０にしておく必要があります。またどの場合でも各レジスタ上位３ビットは無視されます。

5.レジスタR_{11}～R_{12}

エンベロープの周期(**表3-2**のｔに相当する時間)を決定します。R_{11}とR_{12}をペアにして16ビットとして使い，R_{11}に下位８ビット・データ，R_{12}に上位8ビット・データを書き込みます。周期ｔと16ビット・データＤの関係は次の式で与えられます。

$$t = \frac{256 \times D}{fclock}$$

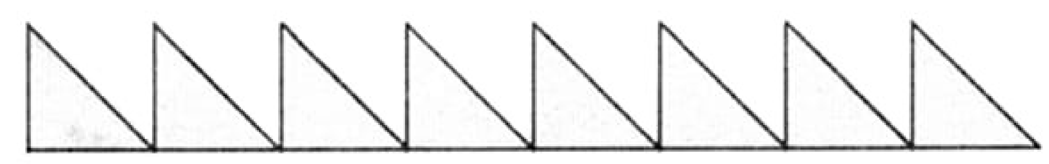
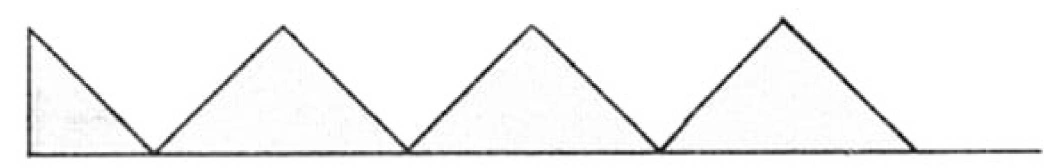
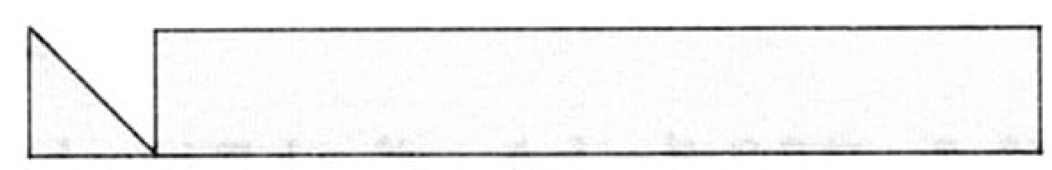
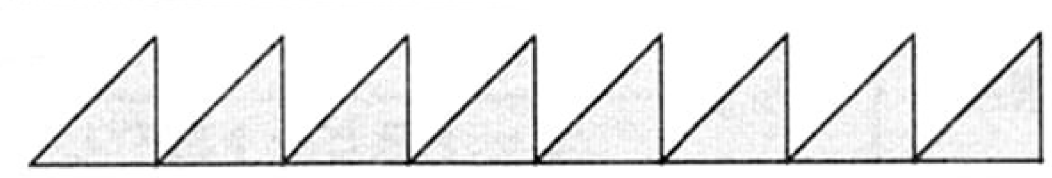
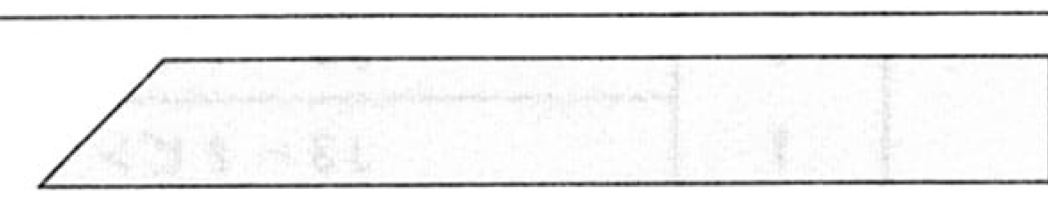
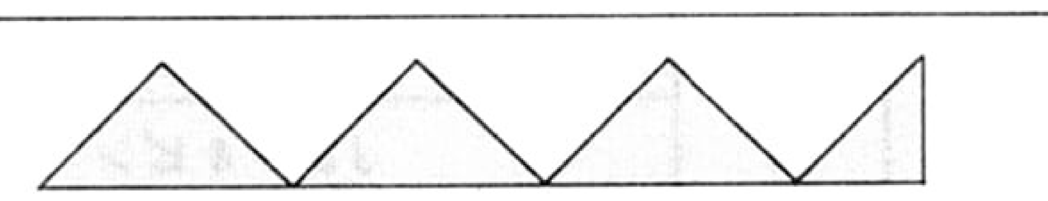
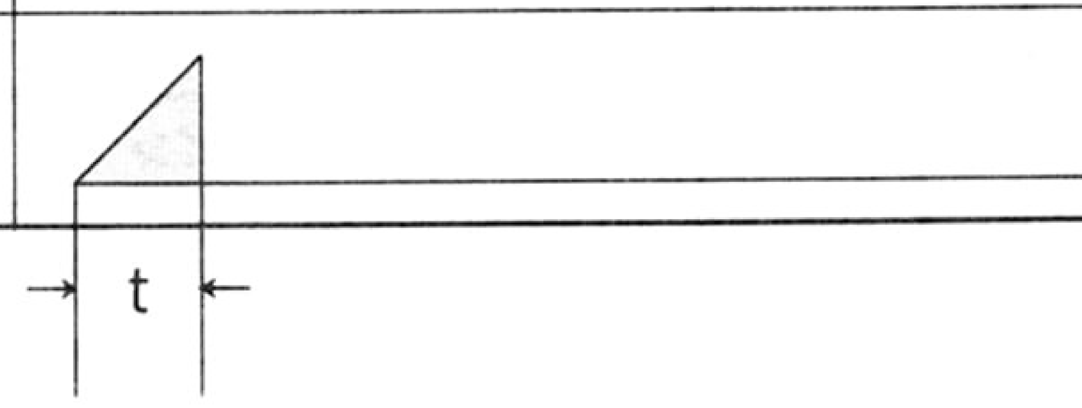
fclock：PSGへのクロック入力（1.7897725MHz）

t　　：エンベロープ周期

D　　：R_{11}とR_{12}に書き込まれた16ビット・データ

MSXの場合には，周期の最少が140マイクロ秒，最大が9.4秒です。

表3-2　エンベロープ・パターン

レジスタR_{13}の下位4ビット	16進値	エンベロープ・パターン
0　0　－　－	0～3	
0　1　－　－	4～7	
1　0　0　0	8	
1　0　0　1	9	
1　0　1　0	A	
1　0　1　1	B	
1　1　0　0	C	
1　1　0　1	D	
1　1　1　0	E	
1　1　1　1	F	

6.レジスタR_{13}

エンベロープのパターンを決定します。パターンの選択は，R_{13}の下位4ビットに書き込まれるデータで**表3-2**のように決まります。R_{13}にデータを書き込むと同時にエンベロープのトリガーがかかるので，再度トリガーをかけたい場合は，そのたびにパターンを指定し直さなければなりません。

※トリガー　もともと引き金の意味で，ここではエンベロープをかけるきっかけとなる信号のこと。

7.レジスタR_{14}～R_{15}

R_{14}，R_{15}とも8ビットの入出力ポートです。これら入出力ポートはサウンド機能とは関係がありません。R_{14}がポートA，R_{15}がポートBで，ＭＳＸではポートAを入力，ポートBを出力として使います。両ポートの各ビットは**表3-3**のように割り当てられています。

ポートA・ビット0～5

ポートAのビット0～7までは，ＭＳＸの汎用入出力ポート（ジョイスティックなどを接続する）のデータを読み取るために使用します，汎用入出力ポートは2つあり，ポートBのビット6で，どちらのポートのデータを読むかを指定します。

ポートA，ビット6

キーボードの配列指定の入力です。このビットが"H"であればJIS配列，"L"であればあいうえお配列です。

ポートA，ビット7

カセットテープのデータ入力用のポートです。

ポートB，ビット0～4

汎用入出力ポートへデータを出力する場合に使用します。ポートBのビット0～3については，このポートを出力用に使用しないとき（ポートAのビット4，5を入力用に使用しているとき）は常に"H"にしておかなくてはなりません。

表3-3　PSGポートA，ポートBビット割り当て

<table>
<tr><th rowspan="2">ポート</th><th rowspan="2">ビット</th><th colspan="2">接続コネクターピン番号</th><th rowspan="2">ジョイスティック使用時の信号</th></tr>
<tr><th>ポートBビット6が"L"のとき</th><th>ポートBビット6が"H"のとき</th></tr>
<tr><td rowspan="8">A（入力）</td><td>0</td><td>J3－1ピン</td><td>J4－1ピン</td><td>ジョイスティックのFWD</td></tr>
<tr><td>1</td><td>J3－2ピン</td><td>J4－2ピン</td><td>ジョイスティックのBACK</td></tr>
<tr><td>2</td><td>J3－3ピン</td><td>J4－3ピン</td><td>ジョイスティックのLEFT</td></tr>
<tr><td>3</td><td>J3－4ピン</td><td>J4－4ピン</td><td>ジョイスティックのRIGHT</td></tr>
<tr><td>4</td><td>J3－6ピン</td><td>J4－6ピン</td><td>ジョイスティックのTRGA</td></tr>
<tr><td>5</td><td>J3－7ピン</td><td>J4－7ピン</td><td>ジョイスティックのTRGB</td></tr>
<tr><td>6</td><td colspan="3">キーボードの配列指定，"H"のときJIS配列，"L"のときあいうえお配列</td></tr>
<tr><td>7</td><td colspan="3">CSAR(カセットテープのリード信号入力)</td></tr>
<tr><td rowspan="8">B（出力）</td><td>0</td><td>J3－6</td><td colspan="2" rowspan="4">出力ポートとして使用しないときは，必らず"H"レベルにしておく</td></tr>
<tr><td>1</td><td>J3－7</td></tr>
<tr><td>2</td><td>J4－6</td></tr>
<tr><td>3</td><td>J4－7</td></tr>
<tr><td>4</td><td>J3－8</td><td colspan="2" rowspan="2"></td></tr>
<tr><td>5</td><td>J4－8</td></tr>
<tr><td>6</td><td colspan="3">ポートAの入力セレクト，"L"のとき汎用ポート1，"H"のとき汎用ポート2</td></tr>
<tr><td>7</td><td colspan="3">KLAMP(カナランプの表示，"L"で点燈)</td></tr>
</table>

ポートB，ビット6

ＭＳＸの２つの汎用入出力ポートのうち，どちらを使うかを指定します。このビットを“Ｌ”にするとポート１，“Ｈ”にするとポート２の入力がポートＡのビット０～５に表われます。

ポートB，ビット7

このビットを“Ｌ”にすると，キーボードのカナ・モード表示ランプが点燈します。

MSXのサウンド機能を使いこなそう

6 PSGへのアクセス

1.CPUとの接続

ＰＳＧにはバス・コントロールのための信号ＢＣ１，ＢＣ２，ＢＤＩＲがあり，これによってデータ・バスに現在どのような信号が乗っているかを知らせます(**表3-4**)。

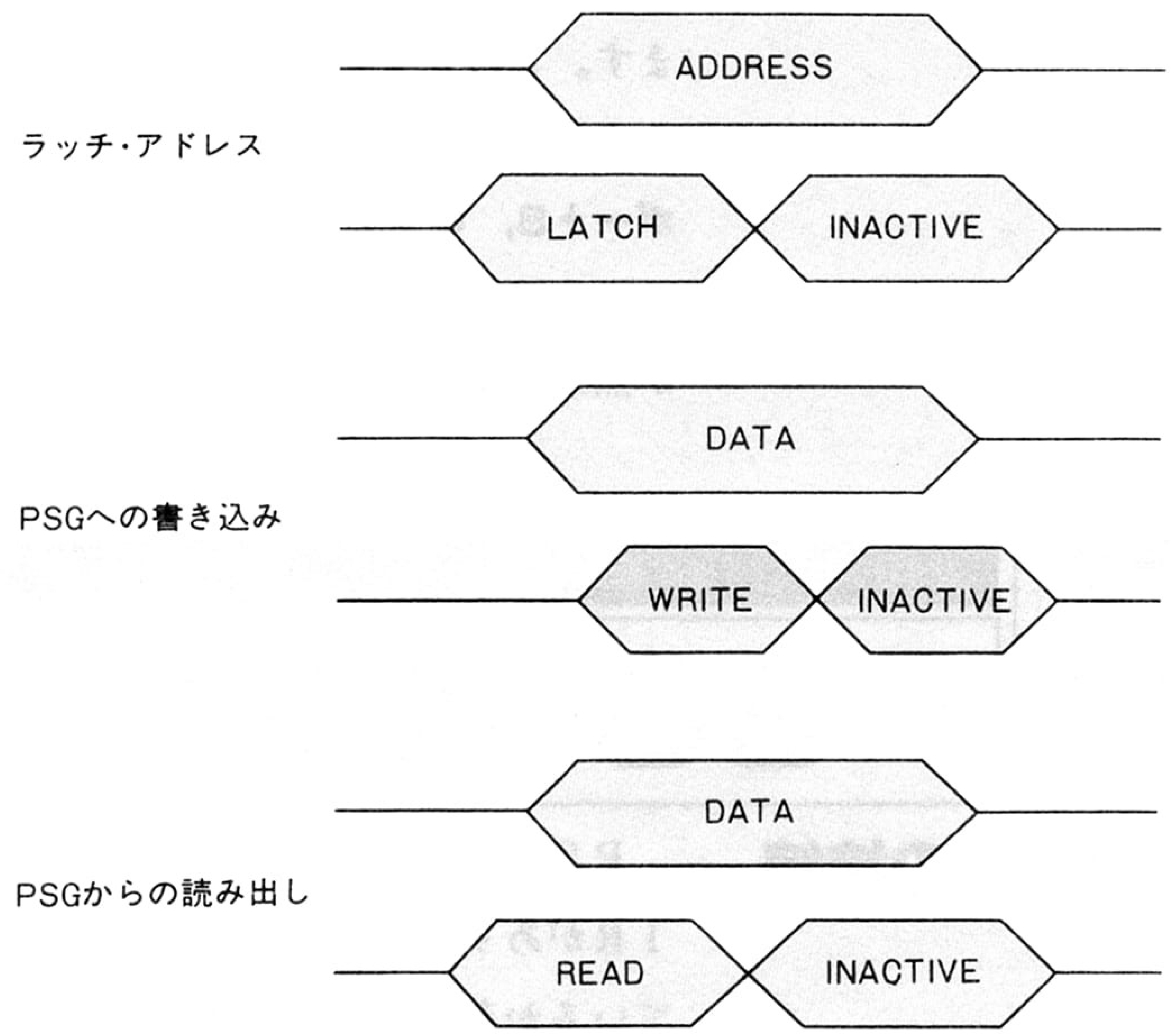
表3-4　データ・バスのコントロール信号

BDIR	BC2	BC1	命　令
0	1	0	インアクティブ
0	1	1	PSGからの読み出し
1	1	0	PSGへの書き込み
1	1	1	ラッチ・アドレス

この表を見てもらうと解かりますが，３本の信号線のうちＢＣ２については常に“Ｈ”レベル固定で使用します。

動作モードには，ラッチ・アドレス（レジスタの指定)，ＰＳＧからの読み出し，ＰＳＧへの書き込み，インアクティブ(動作しない)，の全部で４つがあり，タイミング・チャートはＰ104の**図3-9**のようになります。ハードウェアによるＰＳＧのアクセスは，まずレジスタの指定を行った後，レジスタの読み書きを行います。

図3-9　PSGのタイミング・チャート



ラッチ・アドレス(レジスタの指定)

❶ ラッチ・アドレスの指定(BDIR＝1, BC1＝1)

❷ データ・バスにレジスタの番号を乗せる

❸ インアクティブの指定(BDIR＝0, BC1＝0)

PSGへの書き込み

❶ データ・バスに書き込むデータを乗せる

❷ PSGへの書き込み指定(BDIR＝1, BC1＝0)

❸ インアクティブの指定(BDIR＝0, BC1＝0)

PSGからの読み出し

❶ PSGからの読み出し指定(BDIR＝0, BC1＝1)

❷ データ・バス上のデータを読み込む

❸ インアクティブの指定(BDIR＝0, BC1＝0)

ＭＳＸではこれら一連の動作を行なうために，次のような形でハードウェアが設計されています。

まずアドレス・バスA3～A7をデコードした信号とIORQを使ってチップ・セレクト信号を作ります。さらにアドレス・バスのA0とA1を反転して，それぞれBC1，BDIRに接続します。こうしてPSGは**表3-5**のようにI/Oへ割り当てられます。

表3-5　PSGのI/Oへの割り当て

I/O番号	入　力	出　力
A 0 H	——	PSGのレジスタを指定する
A 1 H	——	PSGのレジスタへデータを書く
A 2 H	PSGのレジスタからデータを読む	——

2.レジスタへの書き込み

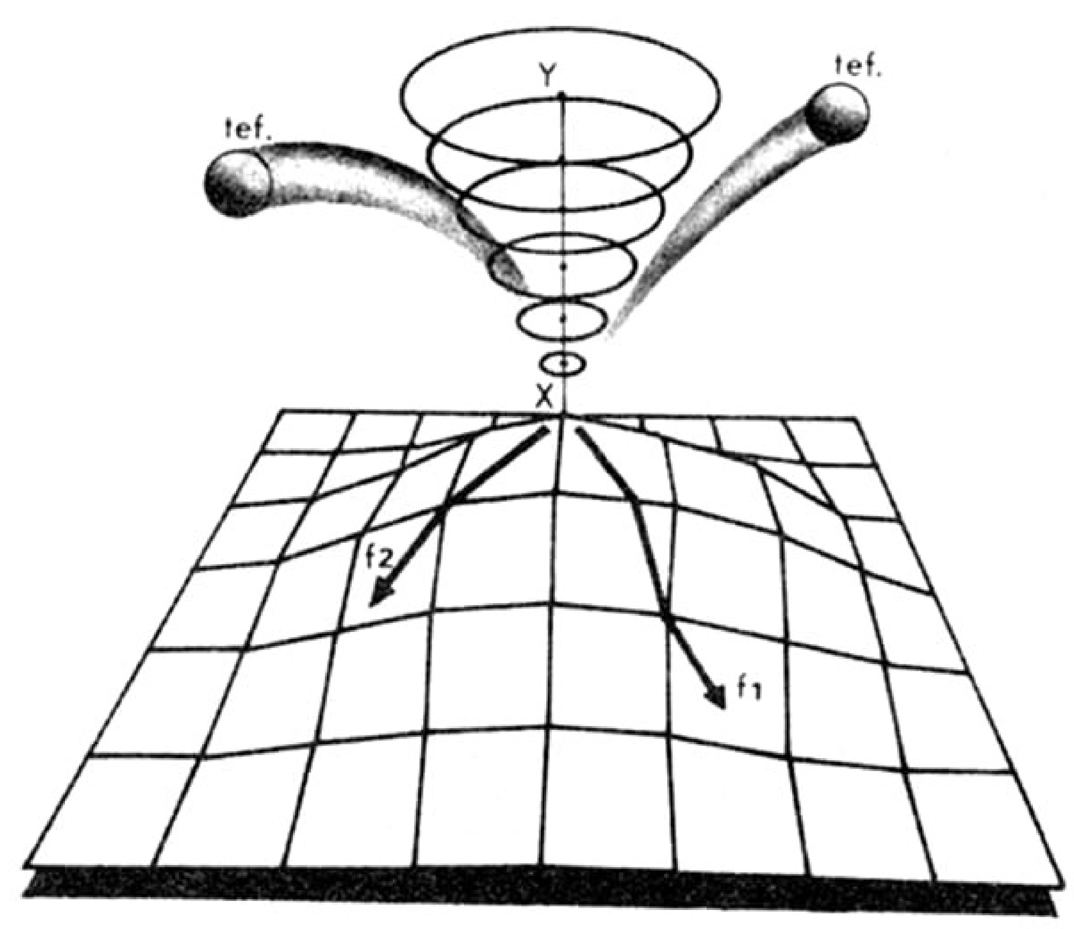
ＭＳＸでのＰＳＧのレジスタへの書き込みは，次のようにします。最初に書き込みを行なうレジスタ番号を，I/OポートＡ０Ｈに出力します。続けてレジスタに書き込む値をポートＡ１Ｈに出力します。レジスタ番号をＣＰＵのＡレジスタ，データをＥレジスタに入れた場合のプログラムを示します。

```
OUT   (0A0H),A………レジスタの指定
LD    A, E
OUT   (0A1H),A………データの書き込み
```

3.レジスタからの読み出し

書き込みの場合と同様，まずポートA0Hに指定するレジスタ番号を出力します。次にポートA2Hからデータを読み取ります。

```
OUT   (0A0H),A………レジスタの指定
IN    A,(0A2H)………データを読み取り
```



MSXのサウンド機能を使いこなす

7 PSGをサポートするBIOSルーチン

ここまでに述べてきたようなＰＳＧのレジスタへの読み書きは，ＭＳＸのBIOSによってサポートされています。したがって今後PSGへのアクセスは，すべてこのBIOSを利用することにします。MSXはポートのレベルまでコンパチビリティと限らないので，特別な場合を除いて直接ＰＳＧをアクセスすることは避けなければなりません。

表3-6はPSGをサポートするBIOSルーチンをまとめたものです。

表3-6　PSGに関するBIOSルーチン

番地	名前	動　作
0090H	GICINI	PSGを初期化し，PLAY文のためスタティック・データをセット
0093H	WRTPSG	PSGのレジスタへデータを書き込む
0096H	RDPSG	PSGのレジスタからデータを読み出す
0099H	STRTMS	PLAY文について，バックグラウンド・タスクをチェックし，スタートさせる

1.GICINI(0090H)

ＰＳＧのレジスタの内容を初期化して，MSX-BASICのPLAY文を実行するための下準備をします。エントリ時のパラメータ，リターン時のパラメータはありません。このルーチンを呼んだ場合，ＣＰＵのすべてのレジスタの内容が変更されます。

2.WRTPSG(0093H)

PSGのレジスタにデータを書き込みます。これはMSX-BASICのSOUND文に相当するものです。CPUのＡレジスタに指定するＰＳＧのレジスタ番号を入れ，Ｅレジスタに書き込みを行うデータを入れます。ＣＰＵのすべてのレジスタの内容は保証されています。

3.RDPSG(0096H)

ＰＳＧのレジスタの内容を読み取ります。Ａレジスタに読み出したいレジスタの番号を入れ，このルーチンをコールすると，Ａレジスタ にＰＳＧのレジスタの内容が返ってきます。Ａレジスタ以外のＣＰＵのレジスタの内容は保証されています。

4.STRTMS(0099H)

バック・グラウンド・タスクとしてPLAY文が実行中であるかどうかを調べ，そうでなければキューにセットされたデータにより，PLAY文の実行を開始します。

MSXのサウンド機能を使いこなす

8 PSGの応用

さてＰＳＧについての一通りの説明を終わったところで，実際にＰＳＧから音を出してみることにしましょう。プログラム中でBIOSルーチンをコールしていますが，これらの動作は前述したとおりです。

1.PSGの初期化

BIOSにはGICINIというPSGの初期化ルーチンが用意されていますが，これはBASICのPLAY文のためのもので，ゲームなどを作る場合には別に初期化ルーチンを用意した方がよいでしょう。

リスト3-1のプログラムでは，チャンネルＡ，Ｂ，Ｃの各トーン出力をＯＮにセットした例です。ただし，このままではまだ音は出力されません。音を出すためには，この他に各チャンネルのトーンの周波数と音量をセットしてやる必要があります。

ところでこれ以後紹介するプログラム例は，事前にＰＳＧの機能がすべてOFFにされていることが前提になっています。したがってそのような場合にこの初期化ルーチンを利用する場合は，レジスタR_7にセットする値をBFH(各チャンネルともトーン，ノイズの出力をOFFにする）に変えます。

リスト3-1

```
 ;
 ;   PROGRAM INITIALIZE PSG
 ;   DATE    6-FEB-84
 ;
          ORG   0D000H
 ;
 ;   BIOS CALL
 ;
 ;   WRTPSG (0093H)
 ;     WRITES TO PSG RESISTER
 ;     ENTRY:   REGISTER# => A
 ;              DATA => E
 ;     RETURNS: NONE
 ;     MODIFIES:NONE
 ;
WRTPSG  EQU   0093H
 ;
 ;   INITIALIZE PSG
 ;
INIPSG  EQU   $
```

```
1200 ';
1210 '          PUSH   AF
1220 '          PUSH   DE
1230 '          PUSH   HL
1240 '          LD     HL,PSGDT ;POINTER:=ADR. OF INITIAL DATA
1250 '          XOR    A        ;REG.#:=0
1260 'INIPS1:  LD     E,(HL)   ;DATA:=(POINTER)
1270 '          CALL   WRTPSG   ;WRITE TO PSG
1280 '          INC    A        ;REG.#:=REG.#+1
1290 '          INC    HL       ;POINTER:=POINTER+1
1300 '          CP     14       ;REG.#=14?
1310 '          JR     NZ,INIPS1;NO,LOOP
1320 '          POP    HL
1330 '          POP    DE
1340 '          POP    AF
1350 '          RET
1360 ';
1370 ';   INITIAL DATA
1380 ';
1390 'PSGDT    EQU    $
1400 ';   REG. R0-R5
1410 '          DEFW   0        ;TONE-A
1420 '          DEFW   0        ;TONE-B
1430 '          DEFW   0        ;TONE-C
1440 ';   REG. R6
1450 '          DEFB   0        ;NOISE FREQUENCY
1460 ';   REG. R7
1470 ';
1480 ';   TONE-A     ON
1490 ';   TONE-B     ON
1500 ';   TONE-C     ON
1510 ';   NOISE-A    OFF
1520 ';   NOISE-B    OFF
1530 ';   NOISE-C    OFF
1540 ';   PORT-A     IN
1550 ';   PORT-B     OUT
1560 '          DEFB   0B8H
1570 ';   REG. R8-R10
1580 '          DEFB   0        ;VOICE-A VOLUME
1590 '          DEFB   0        ;VOICE-B VOLUME
1600 '          DEFB   0        ;VOICE-C VOLUME
1610 ';   REG. R11,R12
1620 '          DEFW   0        ;ENVELOPE FREQUENCY
1630 ';   REG. R13
1640 '          DEFB   0        ;ENVELOPE SHAPE
1650 ';
1660 '          END
```

2.テーマ・ミュージックを演奏する

最近のゲームには，必ずといってよいほどテーマ・ミュージックがあります。アップ・テンポのＢＧＭに誘われて思わず1000円札を両替にいく，なんていうのもしばしば。パックマンなどに代表されるコミカル・タッチのリアルタイム・ゲームでは，このバックに流れる音楽がゲームの面白さに拍車をかけているようです。そこでまずは，ＰＳＧに音楽を演奏させてみることにしましょう。

①任意の周波数の音を出す

さて音楽を演奏させるためには，トーン・ジェネレータの基本的な使い方を知らなければなりません。**リスト3-2**を入力し，アセンブルしたうえで実行させてみてください。スピーカーから楽器のチューニングのときなどに使われる440Hzの基準音(ラの音)が聞こえてくるはずです。もし聞こえてこなかったら，それは機械が故障しているか，あなたに才能がないかのどちらかですので，どちらか一方を修理に出してください。

リスト3-2

```
1000 ';
1010 ';   PROGRAM TEST TONE
1020 ';   DATE    6-FEB-84
1030 ';
1040 '         ORG    0D000H
1050 ';
1060 ';   BIOS CALL
1070 ';
1080 ';   WRTPSG (0093H)
1090 ';     WRITES TO PSG RESISTER
1100 ';     ENTRY:    REGISTER# => A
1110 ';               DATA => E
1120 ';     RETURNS:  NONE
1130 ';     MODIFIES:NONE
1140 ';
1150 'WRTPSG   EQU    0093H
1160 ';
1170 ';   TEST TONE
1180 ';
1190 'TSTTON   EQU    $
1200 ';
1210 '         PUSH   AF
1220 '         PUSH   DE
1230 '         LD     A,7         ;TURN SWITCH
1240 '         LD     E,0BEH      ;TONE-A:=ON
1250 '         CALL   WRTPSG      ;WRITE TO PSG
1260 '         LD     A,8         ;SET VOLUME
```

```
1270 '           LD     E,8H       ;VOLUME-A:=8
1280 '           CALL   WRTPSG     ;WRITE TO PSG
1290 '           LD     A,0        ;SET FREQUENCY
1300 '           LD     E,0FEH     ;TONE-A FREQ.:=440Hz
1310 '           CALL   WRTPSG
1320 '           LD     A,1
1330 '           LD     E,0H
1340 '           CALL   WRTPSG     ;WRITE TO PSG
1350 '
1360 '           POP    DE
1370 '           POP    AF
1380 '           RET
1390 ';
1400 '           END
```

プログラムではチャンネルAのトーン・ジェネレータから440Hzの発振音を出しています。このようにトーン・ジェネレータから音を発生させるには，次のような手順でＰＳＧをセットします。

❶ そのチャンネルのトーン出力をONにする

❷ そのチャンネルの音量をセットする

❸ 発振周波数を計算式に代入して，求まった値をトーン・ジェネレータにセットする。

ここでいう計算式とは以前にも解説した，

$$f=\frac{fclock}{16\times D}$$

のことで，ｆは発振させたい周波数，fclockは1789772.5，Ｄはトーン・ジェネレータに書き込む値になります。したがってf＝440であれば，Ａは次のように求められます。

$$440=\frac{1789772.5}{16\times D} \qquad D\fallingdotseq 254$$

このことからトーン・ジェネレータに254(FEH)をセットすれば，例のような440Hzの発振音が出力されるというわけです。

同様にしてトーン・ジェネレータにデータをセットすれば，任意の周波数の発振音を得ることができます。

②音階の発生

さてトーン・ジェネレータの使用法については，だいたい理解してもらえたと思いますが，このままではいったいどうすればド・レ・ミ…

といった音が出せるのかが解かりません。

人間の耳は高の高さでド・レ・ミ…という区別をします。したがって，ドの音もレの音もそれぞれ個有の周波数を持ちます。そこでそれぞれの音の周波数が解かれば，音階の発生をするのは簡単になります。

表3-7はこの音階と周波数の関係を示したもので，これから，トーン・ジェネレータ用にデータを計算したものが**表3-8**です。

この表さえ手に入れば，あとは一定の間隔で音の長さを測りながら，トーン・ジェネレータにデータを書き込んでいけば音楽の演奏ができます。

表3-7　音階と周波数(Hz)

音階＼オクターブ	1	2	3	4	5	6	7	8
C	32.7	65.4	130.8	261.6	523.2	1046.5	2093.0	4186.0
C＃	34.6	69.3	138.6	277.2	554.4	1108.7	2217.4	4434.9
D	36.7	73.4	146.8	293.7	587.3	1174.7	2349.3	4698.6
D＃	38.9	77.8	155.6	311.1	622.3	1244.5	2489.0	4978.0
E	41.2	82.4	164.8	329.6	659.3	1318.5	2637.0	5274.0
F	43.7	87.3	174.6	349.2	698.5	1396.9	2793.8	5587.6
F＃	46.2	92.5	185.0	370.0	740.0	1480.0	2959.9	5919.9
G	49.0	98.0	196.0	392.0	784.0	1568.0	3135.9	6271.9
G＃	51.9	103.8	207.7	415.3	830.6	1661.2	3322.4	6644.8
A	55.0	110.0	220.0	440.0	880.0	1760.0	3520.0	7040.0
A＃	58.3	116.5	233.1	466.2	932.3	1864.6	3729.3	7458.6
B	61.7	123.5	246.9	493.9	987.8	1975.5	3951.0	7902.1

表3-8　音階データ

音名＼オクターブ	1		2		3		4		5		6		7		8	
	16進	10進	16進	10進	16進	10進	16進	10進	16進	10進	16進	10進	16進	10進	16進	10進
C	D5D	3421	6AF	1711	357	855	1AC	428	D6	214	6B	107	35	53	1B	27
C＃	C9C	3228	64E	1614	327	807	194	404	CA	202	65	101	32	50	19	25
D	BE7	3047	5F4	1524	2FA	762	17D	381	BE	190	5F	95	30	48	18	24
D＃	B3C	2876	59E	1438	2CF	719	168	360	B4	180	5A	90	2D	45	16	22
E	A9B	2715	54E	1358	2A7	679	153	339	AA	170	55	85	2A	42	15	21
F	A02	2562	501	1281	281	641	140	320	A0	160	50	80	28	40	14	20
F＃	973	2419	4BA	1210	25D	605	12E	302	97	151	4C	76	26	38	13	19
G	8EB	2283	476	1142	23B	571	11D	285	8F	143	47	71	24	36	12	18
G＃	86B	2155	436	1078	21B	539	10D	269	87	135	43	67	22	34	11	17
A	7F2	2034	3F9	1017	1FD	509	FE	254	7F	127	40	64	20	32	10	16
A＃	780	1920	3C0	960	1E0	480	F0	240	78	120	3C	60	1E	30	F	15
B	714	1818	38A	906	1C5	453	E3	227	71	113	39	57	1C	28	E	14

③音楽演奏プログラムの使い方

リスト3-3のプログラムは，音楽演奏用のサブルーチンです。演奏させるフレーズは例にあるプログラムのように，インライン・パラメータで指定します。

使い方は次のようになります。

```
CALL  PLAY
DEFW  00D6H……… 1番目の音の高さ
DEFW  0180H ……… 1番目の音の長さ
DEFW  00AAH……… 2番目の音の高さ
DEFW  0080H……… 2番目の音の長さ
DEFW  00BEH……… 3番目の音の高さ
DEFW  0080H ……… 3番目の音の長さ
  ⋮       ⋮
DEFW  0FFFFH……終了のコード
```

音の高さは**表3-8**に示した2バイトの音階データで，長さは**表3-9** に示す同じく2バイトの値の入れます。この長さのデータは♩＝120 のときの値ですから，♩＝nの場合にはこの値を120/n倍してください。休符の場合には音の高さのかわりに0000Hを指定します。

表3-9　音の長さのデータ

音　符	𝅝	𝅗𝅥.	𝅗𝅥	𝅘𝅥.	𝅘𝅥	𝅘𝅥𝅮.	𝅘𝅥𝅮	𝅘𝅥𝅯.	𝅘𝅥𝅯	𝅘𝅥𝅰.	𝅘𝅥𝅰
データ	400H	300H	200H	180H	100H	C0H	80H	60H	40H	30H	20H

♩＝120の場合　　♩＝nの場合はこのデータを$\frac{120}{n}$倍する

※インライン・パラメータ転送

リストを見れば理解できると思いますが，インライン・パラメータ転送というのは，サブルーチンへ分岐する命令の次にパラメータをセットして受け渡す手法のことです。

```
         CALL  INLINE
DATA : DEFB  1, 2, 3
              ⋮
INLINE : EX(SP), HL, またはPOPHL
              ⋮
```

CALL INLINEでスタックには，DATAというラベルのあるアドレスが入ります。INLINEというサブルーチンでスタックからアドレスを得るわけです。この手法は，パラメータが多いときなど非常によく使われます。サブルーチンからの復帰が単純にRETではないことに注意してください。

リスト3-3

```
1000 ';
1010 ';   PROGRAM MUSIC
1020 ';   DATE     10-FEB-84
1030 ';
1040 '          ORG   0D000H
1050 ';
1060 ';   BIOS CALL
1070 ';
1080 ';   WRTPSG (0093H)
1090 ';     WRITES TO PSG RESISTER
1100 ';     ENTRY:    REGISTER# => A
1110 ';               DATA => E
1120 ';     RETURNS: NONE
1130 ';     MODIFIES:NONE
1140 ';
1150 'WRTPSG  EQU   0093H
1160 ';
1170 ';   MUSIC
1180 ';
1190 'MUSIC   EQU   $
1200 ';
1210 '        CALL  PLAY
1220 '        DEFW  017DH      ;NOTE-1 FREQ.
1230 '        DEFW  0040H      ;NOTE-1 LENGTH
1240 '        DEFW  0153H      ;NOTE-2 FREQ.
1250 '        DEFW  0040H      ;NOTE-2 LENGTH
1260 '        DEFW  012EH      ;NOTE-3 FREQ.
1270 '        DEFW  0040H      ;NOTE-3 LENGTH
1280 '        DEFW  0153H      ;NOTE-4 FREQ.
1290 '        DEFW  0040H      ;NOTE-4 LENGTH
1300 '        DEFW  011DH      ;NOTE-5 FREQ.
1310 '        DEFW  0040H      ;NOTE-5 LENGTH
1320 '        DEFW  012EH      ;NOTE-6 FREQ.
1330 '        DEFW  0040H      ;NOTE-6 LENGTH
1340 '        DEFW  0153H      ;NOTE-7 FREQ.
1350 '        DEFW  0040H      ;NOTE-7 LENGTH
1360 '        DEFW  017DH      ;NOTE-8 FREQ.
1370 '        DEFW  0040H      ;NOTE-8 LENGTH
```

```
1380 ';          :      :
1390 ';          :      :
1400 ';          :      :
1410 '          DEFW   0FFFFH      ;END MARK
1420 '          RET
1430 ';
1440 ';   PLAY
1450 ';
1460 'PLAY      EQU    $
1470 ';
1480 '          EX     (SP),HL     ;HL:=ADDRESS OF DATA
1490 '          PUSH   DE
1500 '          PUSH   BC
1510 '          PUSH   AF
1520 '          LD     A,7         ;TURN SWITCH
1530 '          LD     E,0BEH      ;TONE-A:=ON
1540 '          CALL   WRTPSG      ;WRITE TO PSG
1550 'PLY1:     LD     A,8         ;SET VOLUME
1560 '          LD     E,0H        ;VOLUME:=0
1570 '          CALL   WRTPSG      ;WRITE TO PSG
1580 '          LD     C,(HL)      ;GET FREQ. DATA
1590 '          INC    HL
1600 '          LD     B,(HL)
1610 '          INC    HL
1620 '          LD     A,B         ;FREQ.<0?
1630 '          AND    A
1640 '          JP     M,YALP      ;YES,RETURN
1650 '          OR     C           ;FREQ.=0?
1660 '          JR     Z,PLY2      ;YES,VOLUME:=0
1670 '          LD     A,8         ;SET VOLUME
1680 '          LD     E,8H        ;VOLUME:=8
1690 '          CALL   WRTPSG      ;WRITE TO PSG
1700 '          LD     E,C         ;SET FREQ. TO PSG
1710 '          LD     D,B
1720 '          CALL   TONE
1730 'PLY2:     LD     C,(HL)      ;GET LENGTH DATA
1740 '          INC    HL
1750 '          LD     B,(HL)
1760 '          INC    HL
1770 '          CALL   CNTDWN      ;CALL COUNT DOWN
1780 '          JR     PLY1        ;CONTINUE
1790 'YALP:     POP    AF
1800 '          POP    BC
1810 '          POP    DE
1820 '          EX     (SP),HL     ;RECOVER SP
1830 '          RET
1840 ';
1850 ';   TONE
```

```
1860 ';
1870 'TONE     EQU   $
1880 ';
1890 '         XOR   A           ;SET FREQ. TO TONE-A
1900 '         CALL  WRTPSG
1910 '         INC   A
1920 '         LD    E,D
1930 '         CALL  WRTPSG
1940 '         RET
1950 ';
1960 ';   COUNT DOWN
1970 ';
1980 'CNTDWN   EQU   $
1990 ';
2000 'CNT1:    LD    DE,00D8H    ;DELAY
2010 'CNT2:    DEC   DE
2020 '         LD    A,E
2030 '         OR    D
2040 '         JR    NZ,CNT2
2050 '         DEC   BC          ;COUNT DOWN LENGTH DATA
2060 '         LD    A,C
2070 '         OR    B           ;COMPLETE?
2080 '         JR    NZ,CNT1     ;NO,CONTINUE
2090 '         RET
2100 ';
2110 '         END
```

3.エンベロープをかける

音楽の演奏ができたところで，今度はこれにエンベロープをかけてみることにします。エンベロープの効果については口で説明するよりも実際に聴いた方が解かり易いので，実験してその効果を確めてみましょう。

エンベロープを利用する場合，ＰＳＧの設定に次のような手順が加わります。

❶そのチャンネルのエンベロープ・スイッチをONにする（音量の指定は行わない)。

❷　エンベロープ周期の設定を行う。

❸　エンベロープ・パターンを指定する(指定と同時にトリガーがかかるので，必要があれば何度でも指定を行う)。

ここでいうエンベロープ・スイッチは，各チャンネルの音量指定をするレジスタR_8，R_9，R_{10}のビット４(**表3-1**)にあり，このビットをＯＮにするとエンベロープの指定が行われます。

エンベロープ周期については，tを周期，fclockを1789772.5として次の式，

$$t = \frac{256 \times D}{fclock}$$

によって与えられる値Dを，エンベロープ・ジェネレータにセットします。

最後のエンベロープ・パターンですが，これは前提のエンベロープ・パターンの表にあるパターン番号を指定します。音楽演奏の場合は，新たな音を出すたびに再度エンベロープのトリガーをかけてやらなければなりません。したがって新たな音を出すたびにパターンの指定をやり直します。

このようにして先程の音楽演奏ルーチンにパターン番号0のエンベロープをかけてみた例が**リスト3-4**のプログラムです。

リスト3-4

```
1000 ';
1010 ';   PROGRAM PIANO
1020 ';   DATE     10-FEB-84
1030 ';
1040 '         ORG    0D000H
1050 ';
1060 ';   BIOS CALL
1070 ';
1080 ';   WRTPSG (0093H)
1090 ';     WRITES TO PSG RESISTER
1100 ';     ENTRY:    REGISTER# => A
1110 ';               DATA => E
1120 ';     RETURNS: NONE
1130 ';     MODIFIES:NONE
1140 ';
1150 'WRTPSG  EQU    0093H
1160 ';
1170 ';   PIANO
1180 ';
1190 'PIANO   EQU    $
1200 ';
1210 '        CALL   PLAY
1220 '        DEFW   0000H      ;NOTE-1 FREQ.
1230 '        DEFW   0080H      ;NOTE-1 LENGTH
1240 '        DEFW   01ACH      ;NOTE-2 FREQ.
1250 '        DEFW   0080H      ;NOTE-2 LENGTH
```

```
1260 '          DEFW   017DH      ;NOTE-3 FREQ.
1270 '          DEFW   0080H      ;NOTE-3 LENGTH
1280 '          DEFW   0153H      ;NOTE-4 FREQ.
1290 '          DEFW   0080H      ;NOTE-4 LENGTH
1300 '          DEFW   0140H      ;NOTE-5 FREQ.
1310 '          DEFW   0080H      ;NOTE-5 LENGTH
1320 '          DEFW   017DH      ;NOTE-6 FREQ.
1330 '          DEFW   0080H      ;NOTE-6 LENGTH
1340 '          DEFW   0153H      ;NOTE-7 FREQ.
1350 '          DEFW   0080H      ;NOTE-7 LENGTH
1360 '          DEFW   01ACH      ;NOTE-8 FREQ.
1370 '          DEFW   0080H      ;NOTE-8 LENGTH
1380 ';           :      :
1390 ';           :      :
1400 ';           :      :
1410 '          DEFW   0FFFFH     ;END MARK
1420 '          RET
1430 ';
1440 ';    PLAY
1450 ';
1460 'PLAY      EQU    $
1470 ';
1480 '          EX     (SP),HL    ;HL:=ADDRESS OF DATA
1490 '          PUSH   DE
1500 '          PUSH   BC
1510 '          PUSH   AF
1520 '          LD     A,7        ;TURN SWITCH
1530 '          LD     E,0BEH     ;TONE-A:=ON
1540 '          CALL   WRTPSG     ;WRITE TO PSG
1550 '          LD     A,11       ;SET ENVELOPE TIME
1560 '          LD     E,0A8H     ;ENVELOPE TIME:=0.5s
1570 '          CALL   WRTPSG
1580 '          INC    A
1590 '          LD     E,0DH
1600 '          CALL   WRTPSG     ;WRITE TO PSG
1610 'PLY1:     LD     A,8        ;SET VOLUME
1620 '          LD     E,0H       ;VOLUME:=0
1630 '          CALL   WRTPSG     ;WRITE TO PSG
1640 '          LD     C,(HL)     ;GET FREQ. DATA
1650 '          INC    HL
1660 '          LD     B,(HL)
1670 '          INC    HL
1680 '          LD     A,B        ;FREQ.<0?
1690 '          AND    A
1700 '          JP     M,YALP     ;YES,RETURN
1710 '          OR     C          ;FREQ.=0?
1720 '          JR     Z,PLY2     ;YES,VOLUME:=0
1730 '          LD     A,8        ;TURN ENVELOPE SWITCH
```

```
1740 '          LD     E,10H        ;ENVELOPE CHANNEL-A:=ON
1750 '          CALL   WRTPSG       ;WRITE TO PSG
1760 '          LD     A,13         ;SET ENVELOPE SHAPE
1770 '          LD     E,0H         ;ENVELOPE SHAPE:=0
1780 '          CALL   WRTPSG       ;WRITE TO PSG
1790 '          LD     E,C          ;SET FREQ. TO PSG
1800 '          LD     D,B
1810 '          CALL   TONE
1820 'PLY2:     LD     C,(HL)       ;GET LENGTH DATA
1830 '          INC    HL
1840 '          LD     B,(HL)
1850 '          INC    HL
1860 '          CALL   CNTDWN       ;CALL COUNT DOWN
1870 '          JR     PLY1         ;CONTINUE
1880 'YALP:     POP    AF
1890 '          POP    BC
1900 '          POP    DE
1910 '          EX     (SP),HL      ;RECOVER SP
1920 '          RET
1930 ';
1940 ';   TONE
1950 ';
1960 'TONE      EQU    $
1970 ';
1980 '          XOR    A            ;SET FREQ. TO TONE-A
1990 '          CALL   WRTPSG
2000 '          INC    A
2010 '          LD     E,D
2020 '          CALL   WRTPSG
2030 '          RET
2040 ';
2050 ';   COUNT DOWN
2060 ';
2070 'CNTDWN    EQU    $
2080 ';
2090 'CNT1:     LD     DE,00D8H     ;DELAY
2100 'CNT2:     DEC    DE
2110 '          LD     A,E
2120 '          OR     D
2130 '          JR     NZ,CNT2
2140 '          DEC    BC           ;COUNT DOWN LENGTH DATA
2150 '          LD     A,C
2160 '          OR     B            ;COMPLETE?
2170 '          JR     NZ,CNT1      ;NO,CONTINUE
2180 '          RET
2190 ';
2200 '          END
```

4.効果音の合成

PSGはもともとゲームに利用されることを意識して設計されています。したがってPSGにとって効果音の合成は，音楽の演奏と共に重要な機能のとひつです。特にリアルタイム・ゲームにおいては，『バキューン』とか『ドカーン』というような効果音がゲームをより迫力あるものにしてくれます。

①ノイズ・ジェネレータの利用

効果音を合成するにあたって，このノイズ・ジェネレータは必要不可欠の存在です。そこでまずこのノイズ・ジェネレータの使い方を簡単に説明しておきましょう。

ノイズ・ジェネレータを使うためにはPSGを次のような手順でセットします。

❶ 出力したいチャンネルのノイズ出力をONにする。
❷ 出力したいチャンネルの音量をセットする。
❸ ノイズ・ジェネレータにノイズの平均周波数をセットする。

ノイズの平均周波数については，fを周波数，fclockを1789772.5として次の式，

$$f=\frac{fclock}{16\times D}$$

で求められた値Dをノイズ・ジェネレータにセットします。ノイズ・ジェネレータではこのDによって設定された周波数を中心にランダムな発振を行います。

ところでノイズは多くの場合，エンベロープと組み合わせて利用されます。上記の例ではエンベロープのセットを行っていませんが，エンベロープの使い方はトーンの場合とまったく同じです。

ノイズを利用した簡単な例としてピストルの発射音を合成してみました（P120の**リスト3-5**)。これはノイズにピアノと同じ単発のエンベロープをかけたものです。エンベロープ波形を連続したものに取り替えればマシンガンの音になるでしょう。

このプログラムでは，PSGのセットを簡単に行なうためにSOUNDというサブルーチンを使用しています。このサブルーチンは，以下の効果音の合成プログラムで頻繁に利用されるので使い方を説明しておきましょう。サブルーチンSOUNDは，PSGのレジスタに連続してデータを書き込むもので，データは音楽演奏用サブルーチンと同様に次のような形で与えます。

```
CALL  SOUND
DEFB  0    ……レジスタ番号
DEFB  0ACH……書き込むデータ
DEFB  1    ……レジスタ番号
DEFB  1H   ……書き込むデータ
DEFB  2    ……レジスタ番号
  :     :
  :     :
DEFB  0FFH……終了のコード
```

このサブルーチンを使えばPSGに対するデータのセットを効率よく行うことができます。では実際にこのサブルーチンを利用して，効果音を合成した例をいくつか紹介しましょう。

リスト3-5

```
1000 ';
1010 ';   PROGRAM PISTOL
1020 ';   DATE     6-FEB-84
1030 ';
1040 '         ORG    0D000H
1050 ';
1060 ';   BIOS CALL
1070 ';
1080 ';   WRTPSG (0093H)
1090 ';     WRITES TO PSG RESISTER
1100 ';     ENTRY:    REGISTER# => A
1110 ';               DATA => E
1120 ';     RETURNS:  NONE
1130 ';     MODIFIES:NONE
1140 ';
1150 'WRTPSG   EQU    0093H
1160 ';
1170 ';   PISTOL
1180 ';
1190 'PISTOL   EQU    $
1200 ';
1210 '         CALL   SOUND
1220 '         DEFB   7,0B7H      ;NOISE-A:=ON
1230 '         DEFB   6,7H        ;NOISE FREQ.:=15KHz
1240 '         DEFB   8,10H       ;ENVELOPE-A:=ON
1250 '         DEFB   11,0A8H     ;ENVELOPE TIME:=0.5s
1260 '         DEFB   12,0DH
```

```
1270 '          DEFB  13,0        ;ENVELOPE SHAPE:=0
1280 '          DEFB  0FFH        ;END MARK
1290 '          RET
1300 ';
1310 ';   SOUND
1320 ';
1330 'SOUND     EQU   $
1340 ';
1350 '          EX    (SP),HL     ;HL:=ADDRESS OF DATA
1360 '          PUSH  DE
1370 '          PUSH  AF
1380 'SND1:     LD    A,(HL)      ;A:=REGISTER#
1390 '          INC   HL
1400 '          AND   A           ;REGISTER#<0?
1410 '          JP    M,SND2      ;YES,RETURN
1420 '          LD    E,(HL)      ;E:=DATA
1430 '          INC   HL
1440 '          CALL  WRTPSG      ;WRITE TO PSG
1450 '          JR    SND1
1460 'SND2:     POP   AF
1470 '          POP   DE
1480 '          EX    (SP),HL     ;RECOVER SP
1490 '          RET
1500 ';
1510 '          END
```

●さざ波

ホワイト・ノイズに波状のエンベロープをかけて作ります。エンベロープの周期は7，8秒といったところでしょうか。波の音を注意して聞くと，ＰＳＧのエンベロープが直線的でなく，段階的に変化しているのがわかります(**リスト3-6**)。

リスト3-6

```
1000 ';
1010 ';   PROGRAM WAVE
1020 ';   DATE    6-FEB-84
1030 ';
1040 '          ORG   0D000H
1050 ';
1060 ';   BIOS CALL
1070 ';
1080 ';   WRTPSG (0093H)
1090 ';     WRITES TO PSG RESISTER
1100 ';     ENTRY:    REGISTER# => A
1110 ';               DATA => E
1120 ';     RETURNS:  NONE
```

```
1130 ';     MODIFIES:NONE
1140 ';
1150 'WRTPSG  EQU   0093H
1160 ';
1170 ';   WAVE
1180 ';
1190 'WAVE    EQU   $
1200 ';
1210 '        CALL  SOUND
1220 '        DEFB  7,0B7H      ;NOISE-A:=ON
1230 '        DEFB  6,7H        ;NOISE FREQ.:=15KHz
1240 '        DEFB  8,10H       ;ENVELOPE-A:=ON
1250 '        DEFB  11,7AH      ;ENVELOPE TIME:=8s
1260 '        DEFB  12,0DAH
1270 '        DEFB  13,0EH      ;ENVELOPE SHAPE:=14
1280 '        DEFB  0FFH        ;END MARK
1290 '        RET
1300 ';
1310 ';   SOUND
1320 ';
1330 'SOUND   EQU   $
1340 ';
1350 '        EX    (SP),HL     ;HL:=ADDRESS OF DATA
1360 '        PUSH  DE
1370 '        PUSH  AF
1380 'SND1:   LD    A,(HL)      ;A:=REGISTER#
1390 '        INC   HL
1400 '        AND   A           ;REGISTER#<0?
1410 '        JP    M,SND2      ;YES,RETURN
1420 '        LD    E,(HL)      ;E:=DATA
1430 '        INC   HL
1440 '        CALL  WRTPSG      ;WRITE TO PSG
1450 '        JR    SND1
1460 'SND2:   POP   AF
1470 '        POP   DE
1480 '        EX    (SP),HL     ;RECOVER SP
1490 '        RET
1500 ';
1510 '        END
```

●サイレン

ピーポーピーポーというあのサイレンです。これなど2つの周波数のトーンを交互に出力することで，割と簡単に合成することができます(**リスト3-7**)。

リスト3-7

```
1000 ';
1010 ';   PROGRAM SIREN
1020 ';   DATE    6-FEB-84
1030 ';
1040 '           ORG    0D000H
1050 ';
1060 ';   BIOS CALL
1070 ';
1080 ';   WRTPSG (0093H)
1090 ';     WRITES TO PSG RESISTER
1100 ';     ENTRY:    REGISTER# => A
1110 ';               DATA => E
1120 ';     RETURNS: NONE
1130 ';   BREAKX (00B7H)
1140 ';     CHECKS THE STATUS OF
1150 ';     CTRL-STOP KEY
1160 ';     ENTRY:    NONE
1170 ';     RETURNS: IF BEING PRESSED
1180 ';               THEN CY <= 1
1190 ';     MODIFIES:AF
1200 ';
1210 'WRTPSG   EQU    0093H
1220 'BREAKX   EQU    00B7H
1230 ';
1240 ';   SIREN
1250 ';
1260 'SIREN    EQU    $
1270 ';
1280 '         PUSH   AF
1290 '         PUSH   BC
1300 '         CALL   SOUND
1310 '         DEFB   7,0BCH      ;TONE-A:=ON,TONE-B:=ON
1320 '         DEFB   8,10H       ;ENVELOPE-A:=ON
1330 '         DEFB   9,10H       ;ENVELOPE-B:=ON
1340 '         DEFB   11,7AH      ;ENVELOPE TIME:=8s
1350 '         DEFB   12,0DAH
1360 '         DEFB   13,0FEH     ;ENVELOPE SHAPE:=14
1370 '         DEFB   0FFH        ;END MARK
1380 'SIR1:    CALL   SOUND
1390 '         DEFB   0,0D6H      ;TONE-A FREQ.:=523Hz
1400 '         DEFB   1,0
1410 '         DEFB   2,7FH       ;TONE-B FREQ.:=880Hz
1420 '         DEFB   3,0
```

```
1430 '          DEFB  0FFH       ;END MARK
1440 '          CALL  DELY
1450 '          CALL  SOUND
1460 '          DEFB  0,0ACH     ;TONE-A FREQ.:=262Hz
1470 '          DEFB  1,1H
1480 '          DEFB  2,0FEH     ;TONE-B FREQ.:=440Hz
1490 '          DEFB  3,0
1500 '          DEFB  0FFH       ;END MARK
1510 '          CALL  DELY
1520 '          CALL  BREAKX
1530 '          JR    NC,SIR1
1540 '          POP   BC
1550 '          POP   AF
1560 '          RET
1570 ';
1580 ';   DELAY
1590 ';
1600 'DELY      EQU   $
1610 ';
1620 '          LD    BC,8000H
1630 'DEL1:     DEC   BC
1640 '          LD    A,C
1650 '          OR    B
1660 '          JR    NZ,DEL1
1670 '          RET
1680 ';
1690 ';   SOUND
1700 ';
1710 'SOUND     EQU   $
1720 ';
1730 '          EX    (SP),HL    ;HL:=ADDRESS OF DATA
1740 '          PUSH  DE
1750 '          PUSH  AF
1760 'SND1:     LD    A,(HL)     ;A:=REGISTER#
1770 '          INC   HL
1780 '          AND   A          ;REGISTER#<0?
1790 '          JP    M,SND2     ;YES,RETURN
1800 '          LD    E,(HL)     ;E:=DATA
1810 '          INC   HL
1820 '          CALL  WRTPSG     ;WRITE TO PSG
1830 '          JR    SND1
1840 'SND2:     POP   AF
1850 '          POP   DE
1860 '          EX    (SP),HL    ;RECOVER SP
1870 '          RET
1880 ';
1890 '          END
```

●踏み切り

3つの周波数の違うトーンを出力し，そのうち2つにエンベロープをかけて作ります(**リスト3-8**)。

リスト3-8

```
';
';   PROGRAM FUMIKIRI
';   DATE     6-FEB-84
';
'         ORG    0D000H
';
';   BIOS CALL
';
';   WRTPSG (0093H)
';     WRITES TO PSG RESISTER
';     ENTRY:    REGISTER# => A
';               DATA => E
';     RETURNS:  NONE
';     MODIFIES:NONE
';
'WRTPSG   EQU    0093H
';
';   FUMIKIRI
';
'FUMI     EQU    $
';
'         CALL   SOUND
'         DEFB   7,0B8H      ;TONE-A,B,C:=ON
'         DEFB   8,10H       ;ENVELOPE-A:=ON
'         DEFB   9,10H       ;ENVELOPE-B:=ON
'         DEFB   10,0CH      ;VOLUME-C:=12
'         DEFB   0,6BH       ;TONE-A FREQ.:=1047Hz
'         DEFB   1,0
'         DEFB   2,0D6H      ;TONE-B FREQ.:=523Hz
'         DEFB   3,0
'         DEFB   4,0ACH      ;TONE-C FREQ.:=262Hz
'         DEFB   5,1H
'         DEFB   11,0A8H     ;ENVELOPE TIME:=0.5s
'         DEFB   12,0DH
'         DEFB   13,8H       ;ENVELOPE SHAPE:=8
'         DEFB   0FFH        ;END MARK
'         RET
';
';   SOUND
';
'SOUND    EQU    $
';
'         EX     (SP),HL     ;HL:=ADDRESS OF DATA
'         PUSH   DE
```

```
1440 '          PUSH   AF
1450 'SND1:     LD     A,(HL)      ;A:=REGISTER#
1460 '          INC    HL
1470 '          AND    A           ;REGISTER#<0?
1480 '          JP     M,SND2      ;YES,RETURN
1490 '          LD     E,(HL)      ;E:=DATA
1500 '          INC    HL
1510 '          CALL   WRTPSG      ;WRITE TO PSG
1520 '          JR     SND1
1530 'SND2:     POP    AF
1540 '          POP    DE
1550 '          EX     (SP),HL     ;RECOVER SP
1560 '          RET
1570 ';
1580 '          END
```

●爆発音

ゲームの効果音としては最も利用度の高いものです。ピンク・ノイズにエンベロープをかけて作りますが，ＭＳＸではＰＳＧにかける基本クロックの周波数が高いために，ピンク・ノイズのような音を出すことができません。そこでいろいろ苦労した結果，低い周波数のトーンをミックスすることで，何とかそれらしい音に聞かせています（**リスト3-9**）。

リスト3-9

```
1000 ';
1010 ';   PROGRAM EXPLOSION
1020 ';   DATE     6-FEB-84
1030 ';
1040 '          ORG    0D000H
1050 ';
1060 ';   BIOS CALL
1070 ';
1080 ';   WRTPSG (0093H)
1090 ';     WRITES TO PSG RESISTER
1100 ';     ENTRY:    REGISTER# => A
1110 ';               DATA => E
1120 ';     RETURNS:  NONE
1130 ';     MODIFIES:NONE
1140 ';
1150 'WRTPSG   EQU    0093H
1160 ';
1170 ';   EXPLOSION
1180 ';
1190 'EXPLO    EQU    $
```

```
1200 ';
1210 '        CALL   SOUND
1220 '        DEFB   7,0B6H       ;TONE-A:=ON,NOISE-B:=ON
1230 '        DEFB   8,10H        ;ENVELOPE-A:=ON
1240 '        DEFB   0,0FFH       ;TONE-A FREQ.:=27.3Hz
1250 '        DEFB   1,0FH
1260 '        DEFB   6,1FH        ;NOISE FREQ.:=3.6KHz
1270 '        DEFB   11,8CH       ;ENVELOPE TIME:=5s
1280 '        DEFB   12,88H
1290 '        DEFB   13,0         ;ENVELOPE SHAPE:=0
1300 '        DEFB   0FFH         ;END MARK
1310 '        RET
1320 ';
1330 ';   SOUND
1340 ';
1350 'SOUND   EQU    $
1360 ';
1370 '        EX     (SP),HL      ;HL:=ADDRESS OF DATA
1380 '        PUSH   DE
1390 '        PUSH   AF
1400 'SND1:   LD     A,(HL)       ;A:=REGISTER#
1410 '        INC    HL
1420 '        AND    A            ;REGISTER#<0?
1430 '        JP     M,SND2       ;YES,RETURN
1440 '        LD     E,(HL)       ;E:=DATA
1450 '        INC    HL
1460 '        CALL   WRTPSG       ;WRITE TO PSG
1470 '        JR     SND1
1480 'SND2:   POP    AF
1490 '        POP    DE
1500 '        EX     (SP),HL      ;RECOVER SP
1510 '        RET
1520 ';
1530 '        END
```

●急降下爆撃

トーン・ジェネレータの発振周波数を，高い方から低い方へ連続的に変化させ，ポルタメントの効果を得ています。3つのトーンを合成すれば，よりスペイシーで広がりのある音になります(**リスト3-10**)。

リスト3-10

```
1000 ';
1010 ';   PROGRAM BOMBER
1020 ';   DATE     6-FEB-84
1030 ';
1040 '         ORG    0D000H
1050 ';
```

```
1060 ';   BIOS CALL
1070 ';
1080 ';   WRTPSG (0093H)
1090 ';     WRITES TO PSG RESISTER
1100 ';     ENTRY:    REGISTER# => A
1110 ';               DATA => E
1120 ';     RETURNS: NONE
1130 ';     MODIFIES:NONE
1140 ';
1150 'WRTPSG  EQU    0093H
1160 ';
1170 ';   BOMBER
1180 ';
1190 'BOMB    EQU    $
1200 ';
1210 '        PUSH   AF
1220 '        PUSH   BC
1230 '        PUSH   DE
1240 '        PUSH   HL
1250 '        CALL   SOUND
1260 '        DEFB   7,0BEH      ;TONE-A:=ON
1270 '        DEFB   8,0FH       ;VOLUME-A:=15
1280 '        DEFB   0FFH        ;END MARK
1290 '        LD     HL,1H       ;TONE-A FREQ.:=111.9KHz
1300 'BOMB1:  XOR    A           ;DOWNTO 218.9Hz
1310 '        LD     E,L         ;HL:=DATA
1320 '        CALL   WRTPSG
1330 '        INC    A
1340 '        LD     E,H
1350 '        CALL   WRTPSG      ;WRITE TO PSG
1360 '        CALL   DELY        ;DEALAY
1370 '        INC    HL
1380 '        LD     A,H
1390 '        AND    0FEH        ;FREQ:=218.9Hz?
1400 '        JR     Z,BOMB1     ;NO,CONTINUE
1410 '        CALL   SOUND
1420 '        DEFB   7,0B6H      ;TONE-A:=ON,NOISE-B:=ON
1430 '        DEFB   0,0FFH      ;TONE-A FREQ.:=27.3Hz
1440 '        DEFB   1,0FH
1450 '        DEFB   6,1FH       ;NOISE FREQ.:=3.6KHz
1460 '        DEFB   8,10H       ;ENVELOPE-A:=ON
1470 '        DEFB   11,7AH      ;ENVELOPE TIME:=8s
1480 '        DEFB   12,0DAH
1490 '        DEFB   13,0        ;ENVELOPE SHAPE:=0
1500 '        DEFB   0FFH        ;END MARK
1510 '        POP    HL
1520 '        POP    DE
1530 '        POP    BC
```

```
1540 '          POP    AF
1550 '          RET
1560 ';
1570 ';   DELAY
1580 ';
1590 'DELY      EQU    $
1600 ';
1610 '          LD     BC,0400H
1620 'DL1:      DEC    BC
1630 '          LD     A,C
1640 '          OR     B
1650 '          JR     NZ,DL1
1660 '          RET
1670 ';
1680 ';   SOUND
1690 ';
1700 'SOUND     EQU    $
1710 ';
1720 '          EX     (SP),HL    ;HL:=ADDRESS OF DATA
1730 '          PUSH   DE
1740 '          PUSH   AF
1750 'SND1:     LD     A,(HL)     ;A:=REGISTER#
1760 '          INC    HL
1770 '          AND    A          ;REGISTER#<0?
1780 '          JP     M,SND2     ;YES,RETURN
1790 '          LD     E,(HL)     ;E:=DATA
1800 '          INC    HL
1810 '          CALL   WRTPSG     ;WRITE TO PSG
1820 '          JR     SND1
1830 'SND2:     POP    AF
1840 '          POP    DE
1850 '          EX     (SP),HL    ;RECOVER SP
1860 '          RET
1870 ';
1880 '          END
```

●その他の各種効果

PSGはシンセサイザを小さなＬＳＩに組み込んでしまったために，シンセサイザとして本来必要な機能のうち省かれてしまったものもだいぶあります。

もっともだからといってＰＳＧの能力を見限ってしまうのも早計で，そういった機能のほとんどはソフトウェアによってカバーできます。

●エコーをかける

カラオケの必需品といえるのがこのエコーです。もちろんＰＳＧにはエコーをつける機能というのはないので，ソフトウェアでこの効果を実現してみようというわけです。

まずはその原理を説明しましょう。音が物体に反射して返ってくると，そこに時間の遅れを生じ，エコーが生じます。要するにこの遅れをソフトウェアで作ってやれば，それがエコーに聞こえるはずです。

そこでPSGの３つのトーン・ジェネレータを利用して，同じ音楽を少しづつ遅らせて演奏させます。もちろん元の音に対して，反射してきた音は小さいはずなので，音量もだんだん小さくしていきます。こうしてエコーを作ったのが**リスト3-11**のプログラムです。

実際エコーをかけてみると聞いた感じがまったく違ってくるので，一度試してみるといいでしょう。

リスト3-11

```
1000 ';
1010 ';   PROGRAM DELAY
1020 ';   DATE     10-FEB-84
1030 ';
1040 '         ORG    0D000H
1050 ';
1060 ';   BIOS CALL
1070 ';
1080 ';   WRTPSG (0093H)
1090 ';     WRITES TO PSG RESISTER
1100 ';     ENTRY:   REGISTER# => A
1110 ';              DATA => E
1120 ';     RETURNS: NONE
1130 ';     MODIFIES:NONE
1140 ';
1150 'WRTPSG  EQU    0093H
1160 ';
1170 ';   DELAY
1180 ';
1190 'DELAY   EQU    $
1200 ';
1210 '         CALL   PLAY
```

```
1220 '          DEFW   01ACH      ;NOTE-1 FREQ.
1230 '          DEFW   0080H      ;NOTE-1 LENGTH
1240 '          DEFW   017DH      ;NOTE-2 FREQ.
1250 '          DEFW   0080H      ;NOTE-2 LENGTH
1260 '          DEFW   0153H      ;NOTE-3 FREQ.
1270 '          DEFW   0080H      ;NOTE-3 LENGTH
1280 '          DEFW   0140H      ;NOTE-4 FREQ.
1290 '          DEFW   0080H      ;NOTE-4 LENGTH
1300 '          DEFW   011DH      ;NOTE-5 FREQ.
1310 '          DEFW   0080H      ;NOTE-5 LENGTH
1320 '          DEFW   00FEH      ;NOTE-6 FREQ.
1330 '          DEFW   0080H      ;NOTE-6 LENGTH
1340 '          DEFW   00E3H      ;NOTE-7 FREQ.
1350 '          DEFW   0080H      ;NOTE-7 LENGTH
1360 '          DEFW   00D6H      ;NOTE-8 FREQ.
1370 '          DEFW   0080H      ;NOTE-8 LENGTH
1380 ';           :      :
1390 ';           :      :
1400 ';           :      :
1410 '          DEFW   0FFFFH     ;END MARK
1420 '          RET
1430 ';
1440 ';   PLAY
1450 ';
1460 'PLAY      EQU    $
1470 ';
1480 '          EX     (SP),HL    ;HL:=ADDRESS OF DATA
1490 '          PUSH   DE
1500 '          PUSH   BC
1510 '          PUSH   AF
1520 '          PUSH   IX
1530 '          CALL   INIPLY     ;INITIALIZE DELAY
1540 'PLY1:     LD     DE,00D8H
1550 'PLY2:     DEC    DE
1560 '          LD     A,E
1570 '          OR     D
1580 '          JR     NZ,PLY2
1590 '          LD     IX,VOCTBL  ;IX:=VOICE TABLE
1600 '          LD     B,3        ;SET VOICE COUNTER
1610 'PLY3:     LD     A,3        ;SET VOICE
1620 '          SUB    B
1630 '          LD     (VOICE),A
1640 '          LD     E,(IX+CNT) ;DE:=COUNTER
1650 '          LD     D,(IX+CNT+1)
1660 '          LD     A,(IX+CNT+1)
1670 '          AND    A          ;FINISH?
1680 '          JP     P,PLY4     ;NO,CONTINUE
1690 '          LD     A,B
```

```
1700 '          DEC    A          ;FINISH ALL?
1710 '          JR     Z,YALP     ;YES,RETURN
1720 '          JR     PLY5       ;NEXT VOICE
1730 'PLY4:     OR     (IX+CNT)   ;COUNT DOWN COMPLETE?
1740 '          CALL   Z,RDNEXT   ;READ NEXT DATA
1750 '          LD     E,(IX+CNT) ;COUNT DOWN
1760 '          LD     D,(IX+CNT+1)
1770 '          DEC    DE
1780 '          LD     (IX+CNT),E
1790 '          LD     (IX+CNT+1),D
1800 'PLY5:     LD     DE,4       ;COUNT DOWN ALL VOICE?
1810 '          ADD    IX,DE
1820 '          DJNZ   PLY3       ;NO,CONTINUE
1830 '          JR     PLY1
1840 'YALP:     LD     L,(IX+PTR) ;HL:=RETURN ADDRESS
1850 '          LD     H,(IX+PTR+1)
1860 '          POP    IX
1870 '          POP    AF
1880 '          POP    BC
1890 '          POP    DE
1900 '          EX     (SP),HL
1910 '          RET
1920 ';
1930 ';   INITALIZE PLAY
1940 ';
1950 'INIPLY    EQU    $
1960 ';
1970 '          LD     A,7        ;TURN SWITCH
1980 '          LD     E,0B8H     ;TONE-A,B,C:=ON
1990 '          CALL   WRTPSG     ;WRITE TO PSG
2000 '          LD     B,3        ;VOLUME-A,B,C:=0
2010 '          LD     A,8
2020 '          LD     E,0
2030 'INI1:     CALL   WRTPSG
2040 '          INC    A
2050 '          DJNZ   INI1
2060 '          LD     A,3        ;SET VOICE COUNTER
2070 '          LD     IX,VOCTBL  ;IX:=VOICE TABLE
2080 '          LD     BC,DLYTIM  ;BC:=DELAY TIME TABLE
2090 '          LD     DE,4
2100 'INI2:     PUSH   AF
2110 '          LD     A,(BC)     ;SET DELAY TIME
2120 '          INC    BC
2130 '          LD     (IX+CNT),A
2140 '          LD     A,(BC)
2150 '          INC    BC
2160 '          LD     (IX+CNT+1),A
2170 '          LD     (IX+PTR),L ;SET COUNTER
```

```
2180 '         LD    (IX+PTR+1),H
2190 '         ADD   IX,DE       ;INITIALIZE ALL VOICE?
2200 '         POP   AF
2210 '         DEC   A
2220 '         JR    NZ,INI2     ;NO,CONTINUE
2230 '         RET
2240 ';
2250 ';   READ NEXT DATA
2260 ';
2270 'RDNEXT   EQU   $
2280 ';
2290 '         PUSH  HL
2300 '         PUSH  DE
2310 '         PUSH  AF
2320 '         PUSH  IY
2330 '         LD    A,(VOICE) ;SET VOLUME
2340 '         ADD   A,8
2350 '         LD    E,0         ;VOLUME:=0
2360 '         CALL  WRTPSG      ;WRIT TO PSG
2370 '         LD    L,(IX+PTR) ;HL:=DATA POINTER
2380 '         LD    H,(IX+PTR+1)
2390 '         LD    E,(HL)      ;GET FREQ. DATA
2400 '         INC   HL
2410 '         LD    D,(HL)
2420 '         INC   HL
2430 '         LD    A,D
2440 '         AND   A           ;FREQ.<0?
2450 '         JP    M,RDN2      ;YES,DO NOTHING
2460 '         OR    E           ;FREQ.=0?
2470 '         JR    Z,RDN1      ;YES,VOLUME:=0
2480 '         LD    A,(VOICE) ;SET FREQ. TO PSG
2490 '         SLA   A
2500 '         CALL  TONE
2510 '         LD    A,(VOICE) ;CALC. ADR. OF VOLUME DATA
2520 '         LD    E,A
2530 '         LD    D,0
2540 '         LD    IY,VOLUME
2550 '         ADD   IY,DE
2560 '         LD    E,(IY+0)  ;GET VOLUME DATA
2570 '         LD    A,(VOICE) ;SET VOLUME
2580 '         ADD   A,8
2590 '         CALL  WRTPSG      ;WRITE TO PSG
2600 'RDN1:    LD    E,(HL)      ;GET LENGTH DATA
2610 '         INC   HL
2620 '         LD    D,(HL)
2630 '         INC   HL
2640 'RDN2:    LD    (IX+CNT),E ;SET COUNTER
2650 '         LD    (IX+CNT+1),D
```

```
2660 '          LD    (IX+PTR),L ;SAVE DATA POINTER
2670 '          LD    (IX+PTR+1),H
2680 '          POP   IY
2690 '          POP   AF
2700 '          POP   DE
2710 '          POP   HL
2720 '          RET
2730 ';
2740 ';   TONE
2750 ';
2760 'TONE      EQU   $
2770 ';
2780 '          CALL  WRTPSG      ;SET FREQ. TO TONE
2790 '          INC   A
2800 '          LD    E,D
2810 '          CALL  WRTPSG
2820 '          RET
2830 ';
2840 ';   DATA FOR DELAY
2850 ';
2860 'DLYTIM:   DEFW  0000H       ;VOICE-A DELAY TIME
2870 '          DEFW  0040H       ;VOICE-B DELAY TIME
2880 '          DEFW  0080H       ;VOICE-C DELAY TIME
2890 ';
2900 'VOLUME:   DEFB  0FH         ;VOICE-A VOLUME
2910 '          DEFB  0AH         ;VOICE-B VOLUME
2920 '          DEFB  06H         ;VOICE-C VOLUME
2930 ';
2940 ';   WORK AREA
2950 ';
2960 '          ORG   $
2970 ';
2980 'CNT       EQU   0
2990 'PTR       EQU   2
3000 ';
3010 'VOCTBL:   DEFS  12
3020 'VOICE:    DEFS  1
3030 ';
3040 '          END
```

●ビブラートをかける

低い周波数で音にＦＭ変調(Frequency Modulation)をかけると，エレクトーンなどでおなじみのビブラートの効果が得られます(**図3-6a**)ビブラートをかけるときにはLFOを利用する場合が多いのですが，もちろんＰＳＧにはＬＦＯはないのでこれもソフトウェアで作ります。

ＦＭ変調というと一見面倒くさそうですが，実際にはトーン・ジェネレータのデータを一定の周期で増減させてやるだけで基本的に難しいことはありません。ただビブラートの深さと周波数を調節できるようにしたため，一定の周期を作り出す(本来はＬＦＯが担当する)部分の記述が若干長くなりました(**リスト3-12**)。

リスト3-12

```
1000 ';
1010 ';   PROGRAM VIBRATO
1020 ';   DATE     10-FEB-84
1030 ';
1040 '          ORG   0D000H
1050 ';
1060 ';   BIOS CALL
1070 ';
1080 ';   WRTPSG (0093H)
1090 ';     WRITES TO PSG REGISTER
1100 ';     ENTRY:    REGISTER# => A
1110 ';               DATA => E
1120 ';     RETURNS: NONE
1130 ';     MODIFIES:NONE
1140 ';
1150 ';
1160 'WRTPSG  EQU   0093H
1170 ';
1180 ';   VIBRATO
1190 ';
1200 'VIBRAT  EQU   $
1210 ';
1220 '          CALL  PLAY
1230 '          DEFW  01ACH     ;NOTE-1 FREQ.
1240 '          DEFW  0080H     ;NOTE-1 LENGTH
1250 '          DEFW  017DH     ;NOTE-2 FREQ.
1260 '          DEFW  0080H     ;NOTE-2 LENGTH
1270 '          DEFW  0153H     ;NOTE-3 FREQ.
1280 '          DEFW  0080H     ;NOTE-3 LENGTH
1290 '          DEFW  0140H     ;NOTE-4 FREQ.
1300 '          DEFW  0080H     ;NOTE-4 LENGTH
1310 '          DEFW  011DH     ;NOTE-5 FREQ.
1320 '          DEFW  0080H     ;NOTE-5 LENGTH
1330 '          DEFW  00FEH     ;NOTE-6 FREQ.
```

```
1340 '          DEFW  0080H    ;NOTE-6 LENGTH
1350 '          DEFW  00E3H    ;NOTE-7 FREQ.
1360 '          DEFW  0080H    ;NOTE-7 LENGTH
1370 '          DEFW  00D6H    ;NOTE-8 FREQ.
1380 '          DEFW  0080H    ;NOTE-8 LENGTH
1390 ';          :      :
1400 ';          :      :
1410 ';          :      :
1420 '          DEFW  0FFFFH   ;END MARK
1430 '          RET
1440 ';
1450 ';   PLAY
1460 ';
1470 'PLAY      EQU   $
1480 ';
1490 '          EX    (SP),HL  ;HL:=ADDRESS OF DATA
1500 '          PUSH  DE
1510 '          PUSH  BC
1520 '          PUSH  AF
1530 '          LD    A,7      ;TURN SWITCH
1540 '          LD    E,0BEH   ;TONE-A:=ON
1550 '          CALL  WRTPSG   ;WRITE TO PSG
1560 'PLY1:     LD    A,8      ;SET VOLUME
1570 '          LD    E,0H     ;VOLUME:=0
1580 '          CALL  WRTPSG   ;WRITE TO PSG
1590 '          LD    C,(HL)   ;GET FREQ. DATA
1600 '          INC   HL
1610 '          LD    B,(HL)
1620 '          INC   HL
1630 '          LD    (TNFREQ),BC ;SAVE FREQ. DATA
1640 '          LD    A,B
1650 '          AND   A        ;FREQ.<0?
1660 '          JP    M,YALP   ;YES,RETURN
1670 '          OR    C        ;FREQ.=0?
1680 '          JR    Z,PLY2   ;YES,VOLUME:=0
1690 '          LD    A,8      ;SET VOLUME
1700 '          LD    E,8H     ;VOLUME:=8
1710 '          CALL  WRTPSG   ;WRITE TO PSG
1720 '          LD    E,C      ;SET FREQ. TO PSG
1730 '          LD    D,B
1740 '          CALL  TONE
1750 'PLY2:     LD    C,(HL)   ;GET LENGTH DATA
1760 '          INC   HL
1770 '          LD    B,(HL)
1780 '          INC   HL
1790 '          CALL  CNTDWN   ;CALL COUNT DOWN
1800 '          JR    PLY1
1810 'YALP:     POP   AF
```

```
1820 '          POP    BC
1830 '          POP    DE
1840 '          EX     (SP),HL   ;RECOVER SP
1850 '          RET
1860 ';
1870 ';   COUNT DOWN
1880 ';
1890 'CNTDWN    EQU    $
1900 ';
1910 '          PUSH   AF
1920 '          PUSH   BC
1930 '          PUSH   DE
1940 '          PUSH   HL
1950 '          CALL   INILFO    ;INITALIZE LFO
1960 'CNT1:     LD     DE,00D8H ;DELAY
1970 'CNT2:     DEC    DE
1980 '          LD     A,E
1990 '          OR     D
2000 '          JR     NZ,CNT2
2010 '          DEC    BC        ;COUNT DOWN
2020 '          LD     A,(FRQLFO)
2030 '          AND    C         ;OSCILLATE LOW FREQ.?
2040 '          JR     NZ,CNT1   ;NO,CONTINUE
2050 '          LD     HL,(TNFREQ) ;HL:=CURRENT FREQ.
2060 '          LD     DE,(FRQMOD) ;DE:=FREQ. MODULATION
2070 '          LD     A,(UDFLG) ;CHECK FREQ. UP/DOWN
2080 '          INC    A
2090 '          LD     (UDFLG),A
2100 '          AND    03H
2110 '          JP     PE,CNT3   ;UP,THEN TNFREQ+FRQMOD
2120 '          SBC    HL,DE     ;DOWN,THEN TNFRQ-FRQMOD
2130 '          JR     CNT4
2140 'CNT3:     ADD    HL,DE
2150 'CNT4:     LD     (TNFREQ),HL ;SAVE TONE FREQ.
2160 '          EX     DE,HL     ;SET NEW FRQ.
2170 '          CALL   TONE      ;WRITE TO PSG
2180 '          LD     A,C       ;COMPLETE?
2190 '          OR     B
2200 '          JR     NZ,CNT1   ;NO,CONTINUE
2210 '          POP    HL
2220 '          POP    DE
2230 '          POP    BC
2240 '          POP    AF
2250 '          RET
2260 ';
2270 ';   INITIALIZE LFO
2280 ';
2290 'INILFO    EQU    $
```

```
2300 ';
2310 '        XOR    A          ;CLEAR UDFLG
2320 '        LD     (UDFLG),A
2330 '        LD     DE,(TNFREQ) ;CALC. MODULATION DATA
2340 '        LD     A,(VIBDPT)
2350 '        SUB    12
2360 'INILF1: JR     Z,INILF2
2370 '        SRL    D
2380 '        RR     E
2390 '        INC    A
2400 '        JR     INILF1
2410 'INILF2: LD     (FRQMOD),DE ;SET MODULATION DATA
2420 '        LD     E,0FFH     ;CALC. LFO FREQ.
2430 '        LD     A,(FRQRAT)
2440 '        AND    A
2450 'INILF3: JR     Z,INILF4
2460 '        SRL    E
2470 '        DEC    A
2480 '        JR     INILF3
2490 'INILF4: LD     A,E
2500 '        LD     (FRQLFO),A ;SET FREQ. DATA
2510 '        RET
2520 ';
2530 ';   TONE
2540 ';
2550 'TONE    EQU    $
2560 ';
2570 '        XOR    A           ;SET FREQ. TO TONE-A
2580 '        CALL   WRTPSG
2590 '        INC    A
2600 '        LD     E,D
2610 '        CALL   WRTPSG
2620 '        RET
2630 ';
2640 ';    DATA FOR LFO
2650 ';
2660 'FRQRAT: DEFB   4          ;LFO FREQ.<0..8>
2670 'VIBDPT: DEFB   6          ;VIBRATO DEPTH<0..8>
2680 ';
2690 ';    WORK AREA
2700 ';
2710 '        ORG    $
2720 ';
2730 'TNFREQ: DEFS   2
2740 'FRQMOD: DEFS   2
2750 'FRQLFO: DEFS   1
2760 'UDFLG:  DEFS   1
2770 ';
```

```
2780 '          END
```

●トレモロをかける

ビブラートが音をＦＭ変調したのに対し，トレモロは音をＡＭ変調(Ampritude Modulation)させます(**図3-6b**)。ビブラートの場合はLFOをソフトウェアで作り出してやらなければなりませんでしたが，トレモロではエンベロープ・ジェネレータで代用することができます(**リスト3-13**)。

リスト3-13

```
1000 ';
1010 ';   PROGRAM TREMOLO
1020 ';   DATE     10-FEB-84
1030 ';
1040 '          ORG   0D000H
1050 ';
1060 ';   BIOS CALL
1070 ';
1080 ';   WRTPSG (0093H)
1090 ';     WRITES TO PSG RESISTER
1100 ';     ENTRY:   REGISTER# => A
1110 ';              DATA => E
1120 ';     RETURNS: NONE
1130 ';     MODIFIES:NONE
1140 ';
1150 'WRTPSG   EQU   0093H
1160 ';
1170 ';   TREMOLO
1180 ';
1190 'TREMOL   EQU   $
1200 ';
1210 '          CALL  PLAY
1220 '          DEFW  01ACH      ;NOTE-1 FREQ.
1230 '          DEFW  0080H      ;NOTE-1 LENGTH
1240 '          DEFW  017DH      ;NOTE-2 FREQ.
1250 '          DEFW  0080H      ;NOTE-2 LENGTH
1260 '          DEFW  0153H      ;NOTE-3 FREQ.
1270 '          DEFW  0080H      ;NOTE-3 LENGTH
1280 '          DEFW  0140H      ;NOTE-4 FREQ.
1290 '          DEFW  0080H      ;NOTE-4 LENGTH
1300 '          DEFW  011DH      ;NOTE-5 FREQ.
1310 '          DEFW  0080H      ;NOTE-5 LENGTH
1320 '          DEFW  00FEH      ;NOTE-6 FREQ.
1330 '          DEFW  0080H      ;NOTE-6 LENGTH
1340 '          DEFW  00E3H      ;NOTE-7 FREQ.
1350 '          DEFW  0080H      ;NOTE-7 LENGTH
1360 '          DEFW  00D6H      ;NOTE-8 FREQ.
```

```
1370 '          DEFW  0080H      ;NOTE-8 LENGTH
1380 ';           :     :
1390 ';           :     :
1400 ';           :     :
1410 '          DEFW  0FFFFH     ;END MARK
1420 '          RET
1430 ';
1440 ';    PLAY
1450 ';
1460 'PLAY      EQU   $
1470 ';
1480 '          EX    (SP),HL    ;HL:=ADDRESS OF DATA
1490 '          PUSH  DE
1500 '          PUSH  BC
1510 '          PUSH  AF
1520 '          LD    A,7        ;TURN SWITCH
1530 '          LD    E,0BEH     ;TONE-A:=ON
1540 '          CALL  WRTPSG     ;WRITE TO PSG
1550 '          LD    A,11       ;SET ENVELOPE TIME
1560 '          LD    E,5EH      ;ENVELOPE TIME:=0.05s
1570 '          CALL  WRTPSG
1580 '          INC   A
1590 '          LD    E,1H
1600 '          CALL  WRTPSG     ;WRITE TO PSG
1610 'PLY1:     LD    A,8        ;SET VOLUME
1620 '          LD    E,0H       ;VOLUME:=0
1630 '          CALL  WRTPSG     ;WRITE TO PSG
1640 '          LD    C,(HL)     ;GET FREQ. DATA
1650 '          INC   HL
1660 '          LD    B,(HL)
1670 '          INC   HL
1680 '          LD    A,B        ;FREQ.<0?
1690 '          AND   A
1700 '          JP    M,YALP     ;YES,RETURN
1710 '          OR    C          ;FREQ.=0?
1720 '          JR    Z,PLY2     ;YES,VOLUME:=0
1730 '          LD    A,8        ;TURN ENVELOPE SWITCH
1740 '          LD    E,10H      ;ENVELOPE CHANNEL-A:=ON
1750 '          CALL  WRTPSG     ;WRITE TO PSG
1760 '          LD    A,13       ;SET ENVELOPE SHAPE
1770 '          LD    E,8H       ;ENVELOPE SHAPE:=8
1780 '          CALL  WRTPSG     ;WRITE TO PSG
1790 '          LD    E,C        ;SET FREQ. TO PSG
1800 '          LD    D,B
1810 '          CALL  TONE
1820 'PLY2:     LD    C,(HL)     ;GET LENGTH DATA
1830 '          INC   HL
1840 '          LD    B,(HL)
```

```
1850 '          INC    HL
1860 '          CALL   CNTDWN     ;CALL COUNT DOWN
1870 '          JR     PLY1       ;CONTINUE
1880 'YALP:     POP    AF
1890 '          POP    BC
1900 '          POP    DE
1910 '          EX     (SP),HL    ;RECOVER SP
1920 '          RET
1930 ';
1940 ';   TONE
1950 ';
1960 'TONE      EQU    $
1970 ';
1980 '          XOR    A          ;SET FREQ. TO TONE-A
1990 '          CALL   WRTPSG
2000 '          INC    A
2010 '          LD     E,D
2020 '          CALL   WRTPSG
2030 '          RET
2040 ';
2050 ';   COUNT DOWN
2060 ';
2070 'CNTDWN    EQU    $
2080 ';
2090 'CNT1:     LD     DE,00D8H   ;DELAY
2100 'CNT2:     DEC    DE
2110 '          LD     A,E
2120 '          OR     D
2130 '          JR     NZ,CNT2
2140 '          DEC    BC         ;COUNT DOWN LENGTH DATA
2150 '          LD     A,C
2160 '          OR     B          ;COMPLETE?
2170 '          JR     NZ,CNT1    ;NO,CONTINUE
2180 '          RET
2190 ';
2200 '          END
```

●ポルタメント

ある高さの音から別の高さの音まで，連続的に周波数が変化するときに得られる効果をいいます(**図3-6c**)。さきほどの『急降下爆撃』はこのポルタメントを使ったよい例です。

原理としては，2つの周波数の間を適当に分割して，段階的に変化させてやればよいので，割と簡単に作ることができます。分割する間隔をせまくしてやれば，より滑らかに変化するようになります（**リスト3-14**)。

リスト3-14

```
1000 ';
1010 ';   PROGRAM PORTAMENTO
1020 ';   DATE     10-FEB-84
1030 ';
1040 '         ORG    0D000H
1050 ';
1060 ';   BIOS CALL
1070 ';
1080 ';   WRTPSG (0093H)
1090 ';     WRITES TO PSG REGISTER
1100 ';     ENTRY:     REGISTER# => A
1110 ';                DATA => E
1120 ';     RETURNS:  NONE
1130 ';     MODIFIES:NONE
1140 ';
1150 ';
1160 'WRTPSG   EQU    0093H
1170 ';
1180 ';   PORTAMENTO
1190 ';
1200 'PORTA    EQU    $
1210 ';
1220 '         CALL   PLAY
1230 '         DEFW   01ACH       ;NOTE-1 FREQ.
1240 '         DEFW   0080H       ;NOTE-1 LENGTH
1250 '         DEFW   017DH       ;NOTE-2 FREQ.
1260 '         DEFW   0080H       ;NOTE-2 LENGTH
1270 '         DEFW   0153H       ;NOTE-3 FREQ.
1280 '         DEFW   0080H       ;NOTE-3 LENGTH
1290 '         DEFW   0140H       ;NOTE-4 FREQ.
1300 '         DEFW   0080H       ;NOTE-4 LENGTH
1310 '         DEFW   011DH       ;NOTE-5 FREQ.
1320 '         DEFW   0080H       ;NOTE-5 LENGTH
1330 '         DEFW   00FEH       ;NOTE-6 FREQ.
1340 '         DEFW   0080H       ;NOTE-6 LENGTH
1350 '         DEFW   00E3H       ;NOTE-7 FREQ.
1360 '         DEFW   0080H       ;NOTE-7 LENGTH
1370 '         DEFW   00D6H       ;NOTE-8 FREQ.
1380 '         DEFW   0080H       ;NOTE-8 LENGTH
```

```
1390 ';           :       :
1400 ';           :       :
1410 ';           :       :
1420 '          DEFW   0FFFFH      ;END MARK
1430 '          RET
1440 ';
1450 ';    PLAY
1460 ';
1470 'PLAY      EQU    $
1480 ';
1490 '          EX     (SP),HL     ;HL:=ADDRESS OF DATA
1500 '          PUSH   DE
1510 '          PUSH   BC
1520 '          PUSH   AF
1530 '          LD     DE,0        ;CLEAR OLD FREQUENCY
1540 '          LD     (OLDFRQ),DE
1550 '          LD     A,7         ;TURN SWITCH
1560 '          LD     E,0BEH      ;TONE-A:=ON
1570 '          CALL   WRTPSG      ;WRITE TO PSG
1580 'PLY1:     LD     A,8         ;SET VOLUME
1590 '          LD     E,0H        ;VOLUME:=0
1600 '          CALL   WRTPSG      ;WRITE TO PSG
1610 '          LD     C,(HL)      ;GET FREQ. DATA
1620 '          INC    HL
1630 '          LD     B,(HL)
1640 '          INC    HL
1650 '          LD     (TNFREQ),BC ;SAVE FREQ. DATA
1660 '          LD     A,B
1670 '          AND    A           ;FREQ.<0?
1680 '          JP     M,YALP      ;YES,RETURN
1690 '          OR     C           ;FREQ.=0?
1700 '          JR     Z,PLY2      ;YES,VOLUME:=0
1710 '          LD     A,8         ;SET VOLUME
1720 '          LD     E,8H        ;VOLUME:=8
1730 '          CALL   WRTPSG      ;WRITE TO PSG
1740 '          LD     E,C         ;SET FREQ. TO PSG
1750 '          LD     D,B
1760 '          CALL   TONE
1770 'PLY2:     LD     C,(HL)      ;GET LENGTH DATA
1780 '          INC    HL
1790 '          LD     B,(HL)
1800 '          INC    HL
1810 '          CALL   CNTDWN      ;CALL COUNT DOWN
1820 '          JR     PLY1
1830 'YALP:     POP    AF
1840 '          POP    BC
1850 '          POP    DE
1860 '          EX     (SP),HL     ;RECOVER SP
1870 '          RET
1880 ';
1890 ';    COUNT DOWN
```

```
1900 ';
1910 'CNTDWN   EQU    $
1920 ';
1930 '         PUSH   AF
1940 '         PUSH   BC
1950 '         PUSH   DE
1960 '         PUSH   HL
1970 '         CALL   INIPRT     ;INITIALIZE PORTAMENTO
1980 'CNT1:    LD     DE,00D8H   ;DELAY
1990 'CNT2:    DEC    DE
2000 '         LD     A,E
2010 '         OR     D
2020 '         JR     NZ,CNT2
2030 '         DEC    BC         ;COUNT DOWN
2040 '         LD     A,(PRTDPT)
2050 '         AND    C          ;PORTAMENTO?
2060 '         JR     NZ,CNT1    ;NO,CONTINUE
2070 '         LD     A,(PRTCNT)
2080 '         AND    A          ;PORT. DONE?
2090 '         JR     Z,CNT5     ;YES,NO MODULATION
2100 '         PUSH   BC         ;SAVE COUNTER
2110 '         LD     HL,(OLDFRQ)  ;OLD
                                    FREQ.:=OLD FREQ.+MOD FREQ.
2120 '         LD     DE,(FRQMD1)  ;CALC. MOD FREQ.
2130 '         ADD    HL,DE
2140 '         LD     A,(FRQMD2)
2150 '         LD     C,A
2160 '         LD     A,(PRTCNT)
2170 '         LD     DE,0
2180 'CNT3:    SRL    A
2190 '         JR     C,CNT4
2200 '         SRL    C
2210 '         ADC    HL,DE
2220 '         JR     CNT3
2230 'CNT4:    LD     A,(PRTCNT)  ;DECREMENT PORT. COUNTER
2240 '         DEC    A
2250 '         LD     (PRTCNT),A
2260 '         LD     (OLDFRQ),HL  ;SAVE OLD FREQ.
2270 '         EX     DE,HL
2280 '         XOR    A          ;SET NEW FREQ. TO PSG
2290 '         CALL   TONE
2300 '         POP    BC         ;RECOVER COUNTER
2310 'CNT5:    LD     A,C        ;COMPLETE?
2320 '         OR     B
2330 '         JP     NZ,CNT1    ;NO,CONTINUE
2340 '         POP    HL
2350 '         POP    DE
2360 '         POP    BC
2370 '         POP    AF
2380 '         RET
2390 ';
```

```
2400 '; INITIALIZE PORT.
2410 ';
2420 'INIPRT  EQU   $
2430 ';
2440 '        PUSH  BC
2450 '        LD    HL,(TNFREQ) ;MOD FREQ.:=(TNFREQ-OLDFREQ)
2460 '        LD    DE,(OLDFRQ) ;/16
2470 '        LD    A,E         ;OLDFRQ=0?
2480 '        OR    D
2490 '        JR    NZ,INIPR1
2500 '        LD    DE,(TNFREQ) ;YES,OLDFRQ:=TNFREQ
2510 '        LD    (OLDFRQ),DE
2520 'INIPR1: SBC   HL,DE       ;CALC. MOD FREQ.
2530 '        LD    B,4
2540 '        XOR   A
2550 'INIPR2: SRA   H
2560 '        RR    L
2570 '        RL    A
2580 '        DJNZ  INIPR2
2590 '        LD    (FRQMD1),HL ;SAVE MOD FREQ.
2600 '        LD    (FRQMD2),A
2610 '        LD    E,0
2620 '        LD    A,(FRQRAT) ;CALC. PORT. DEPTH
2630 '        AND   A
2640 'INIPR4: JR    Z,INIPR5
2650 '        SLA   E
2660 '        INC   E
2670 '        DEC   A
2680 '        JR    INIPR4
2690 'INIPR5: LD    A,E
2700 '        LD    (PRTDPT),A ;SAVE PORT. DEPTH
2710 '        LD    A,10H       ;PORT. COUNTER:=16
2720 '        LD    (PRTCNT),A
2730 '        POP   BC
2740 '        RET
2750 ';
2760 ';  TONE
2770 ';
2780 'TONE    EQU   $
2790 ';
2800 '        XOR   A           ;SET FREQ. TO TONE-A
2810 '        CALL  WRTPSG
2820 '        INC   A
2830 '        LD    E,D
2840 '        CALL  WRTPSG
2850 '        RET
2860 ';
2870 ';   DATA FOR PORTAMENT
2880 ';
2890 'FRQRAT: DEFB  1           ;PORT. DIPTH
2900 ';
```

```
2910 ';    WORK AREA
2920 ';
2930 '        ORG    $
2940 ';
2950 'TNFREQ: DEFS   2
2960 'OLDFRQ: DEFS   2
2970 'FRQMD1: DEFS   2
2980 'FRQMD2: DEFS   1
2990 'PRTCNT: DEFS   1
3000 'PRTDPT: DEFS   1
3010 ';
3020 '        END
```

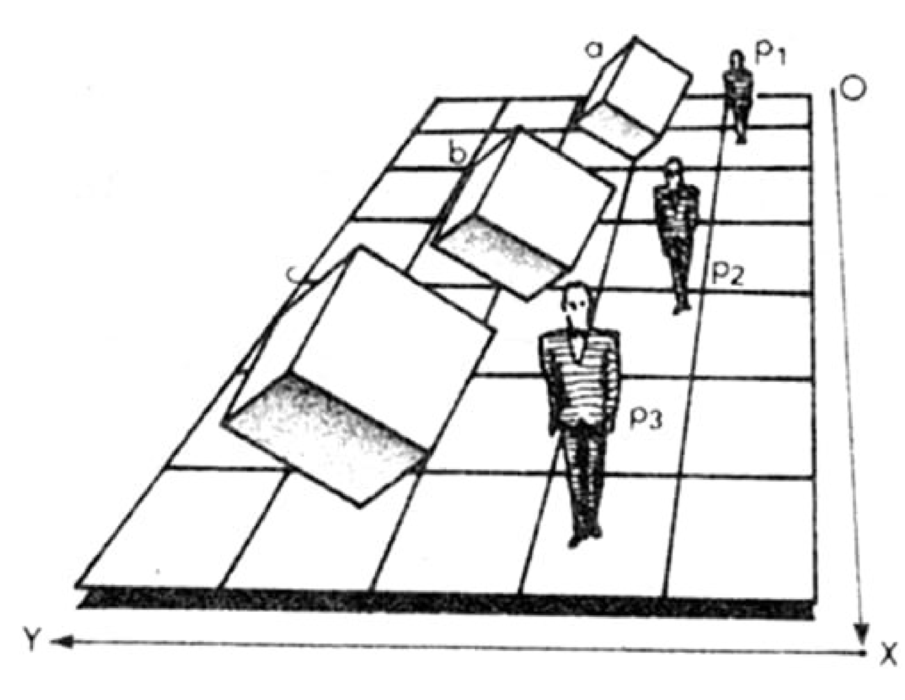
MSXのサウンド機能を使いこなす

9 PLAY文コンパイラ

これまで紹介してきた音楽演奏プログラムは，PSGの3つのトーン・ジェネレータのうち，1つのチャンネルしか利用していなかったので，和音を出したりすることができません。また，CPUが音楽の演奏にかかりっきりになってしまい，音楽演奏に合わせてグラフィック表示をすることもできません。

MSX-BASICではPLAY文という音楽演奏用のステートメントがあり，これは演奏のためにインターバル・タイマによる割り込みを利用しているので，画面表示と音楽演奏を同時に行うことができます。またミュージック・マクロ言語という音楽用の簡易言語が用意されている他，3声までの同時出力がサポートされています。

マシン語でプログラムを組む場合にも，こうしたPLAY文のような機能があれば大変便利です。BIOSにはPLAY文のためのルーチンが用意されていますが，使い方が公開されていません。そこでこれらルーチンの活用を考え，ゲーム開発ツールとしてPLAY文のコンパイラを作ったので紹介します。



1.PLAY文コンパイラの原理

まずこのPLAY文コンパイラの動作原理について解説しましょう。

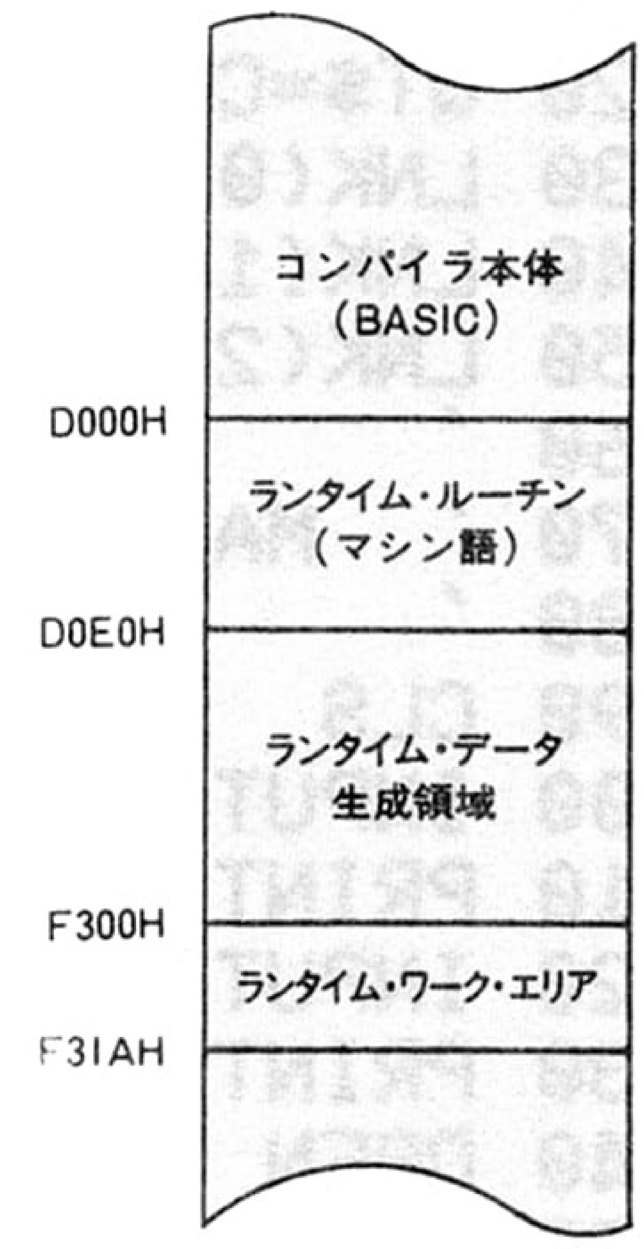
MSX-BASICは，実行時にPLAY文を見つけると続く文字列の中にあるミュージック・マクロ言語を解析し，中間コードのような形に変換してキュー(F975H)に入れます。あとはインターバル・タイマ(VDPによる1/60秒ごとの割り込み)を利用して，キューの中身を解析しながら演奏するのです。

そこでまずPLAY文を実行した直後にキューの中身を取り出して，メモリの他の部分に保存しておきます。あとでマシン語プログラムにより，保存しておいたデータをキューの中に送り込んでやれば，音楽の演奏が再現できます。つまりマシン語でPLAY文を実行することができるというわけです。

2.PLAY文コンパイラの構成

PLAY文コンパイラは，BASICで記述されたコンパイラ本体と実行時に使用するマシン語のランタイム・ルーチンの2つからなります(**リスト3-15, リスト3-16**)

図3-9　PLAY文コンパイラ・メモリ・マップ



3.コンパイラ本体

コンパイラ本体はPLAY文を実行した後キューの中身を取り出し，データとしてメモリに格納します。

コンパイラ本体を実行させるとまずファイル名を聞いてくるので，事前にアスキーセーブしておいたソース・プログラムのファイル名を入力します。このソース・プログラムの作り方についてはあとで解説します。

次に，データをどの番地から格納するのか尋ねてくるので，それを16進数で入力します。ここでただリターンを入力するとデータの先頭はD0E0Hに設定されます。あとはコンパイラがカセットからソース・プログラムを読みながら，データを順に生成していきます。コンパイルが無事終了すると各チャンネルのデータの先頭アドレスとデータの終了アドレスを表示して，BASICのコマンド・モードに戻ります。

リスト3-15

```
1000 '
1010 '   PROGRAM MML COMPILER
1020 '   DATE    1-MAR-84
```

```
1030
1040 CLEAR 300,&HCFFF
1050 MAXFILES=1
1060 DIM LNK(2),M$(2)
1070 DEF FND(I)=PEEK(I)+PEEK(I+1)*256
1080 DEF FNL(I)=I-INT(I/256)*256
1090 DEF FNH(I)=INT(256+I/256)
1100 ON ERROR GOTO 2710
1110 '
1120 '  INTERFACE AREA
1130 '
1140 QUETAB=&HF959
1150 VOICAQ=&HF975
1160 PLYCNT=&HFB40
1170 SEQWRK=&HF300
1180 DTADDR=&HD0E0
1190 '
1200 '  INITIALIZE
1210 '
1220 QT$=CHR$(34)
1230 LNK(0)=SEQWRK
1240 LNK(1)=SEQWRK+2
1250 LNK(2)=SEQWRK+4
1260 '
1270 '  MAIN
1280 '
1290 CLS
1300 INPUT"SOURCE FILE = ";F$
1310 PRINT
1320 INPUT"CODE ADDRESS = ";A$
1330 PRINT
1340 OPEN "CAS:"+F$ FOR INPUT AS #1
1350 IF A$="" THEN PTR=DTADDR ELSE PTR=VAL("&H"+A$)
1360 IF EOF(1) THEN 1410
1370  LINE INPUT #1,L$
1380  PRINT L$
1390  GOSUB 1520
1400 GOTO 1360
1410 CLOSE #1
1420 PRINT
1430 PRINT"VOICE-A : ";HEX$(FND(SEQWRK))
1440 PRINT"VOICE-B : ";HEX$(FND(SEQWRK+2))
1450 PRINT"VOICE-C : ";HEX$(FND(SEQWRK+4))
1460 PRINT
1470 PRINT"CODE END = ";HEX$(PTR)
1480 END
1490 '
1500 '  COMPILE ONE LINE
```

```
1510 '
1520 L$=L$+"@"
1530 LP=0
1540 GOSUB 1830
1550 GOSUB 1900
1560 GOSUB 2000
1570 IF LEFT$(W$,3)="REM" THEN 1790
1580 IF W$<>"PLAY" THEN 2660
1590 M$(0)=""
1600 M$(1)=""
1610 M$(2)=""
1620 FOR V=0 TO 2
1630  GOSUB 2120
1640  M$(V)=S$
1650  IF S$="!"  THEN 2660
1660  IF C$="@" OR V=2 THEN 1700
1670  IF C$<>"," THEN 2660
1680  GOSUB 1830
1690 NEXT
1700 GOSUB 2260
1710 PLAY M$(0),M$(1),M$(2)
1720 IF PLAY(0) THEN 1720
1730 FOR V=0 TO 2
1740  GOSUB 2350
1750  GOSUB 2410
1760  GOSUB 2480
1770  GOSUB 2560
1780 NEXT
1790 RETURN
1800 '
1810 '   GET ONE CHARACTER
1820 '
1830  LP=LP+1
1840  C$=MID$(L$,LP,1)
1850 IF C$=" " THEN 1830
1860 RETURN
1870 '
1880 '   GET NUMBER
1890 '
1900 N$=""
1910 IF C$<"0" OR C$>"9" THEN 1960
1920  N$=N$+C$
1930  NUM=VAL(N$)
1940  GOSUB 1830
1950 GOTO 1910
1960 RETURN
1970 '
1980 '   GET RESERVED WORD
```

```
1990 '
2000 W$=""
2010 IF C$<"A" OR C$>"Z" THEN 2050
2020  W$=W$+C$
2030  GOSUB 1830
2040 GOTO 2010
2050 IF W$<>"" OR C$<>"'" THEN 2080
2060  W$="REM"
2070  GOSUB 1830
2080 RETURN
2090 '
2100 '   GET STRINGS
2110 '
2120 IF C$=QT$ THEN 2150
2130  S$="!"
2140 GOTO 2220
2150  GOSUB 1830
2160  S$=""
2170  IF C$=QT$ OR C$="@" THEN 2210
2180   S$=S$+C$
2190   GOSUB 1830
2200  GOTO 2170
2210  IF C$=QT$ THEN GOSUB 1830
2220 RETURN
2230 '
2240 '   INITIALIZE QUEUE
2250 '
2260 FOR T=0 TO 2
2270  POKE QUETAB+T*6,0
2280  POKE QUETAB+T*6+1,0
2290 NEXT
2300 POKE PLYCNT,0
2310 RETURN
2320 '
2330 '   SET LINK
2340 '
2350 POKE LNK(V),FNL(PTR)
2360 POKE LNK(V)+1,FNH(PTR)
2370 RETURN
2380 '
2390 '   PUT SIZE
2400 '
2410 MLN=PEEK(QUETAB+V*6)+1
2420 POKE PTR,MLN
2430 PTR=PTR+1
2440 RETURN
2450 '
2460 '   PUT LINK
```

```
2470 '
2480 LNK(V)=PTR
2490 POKE PTR,0
2500 POKE PTR+1,0
2510 PTR=PTR+2
2520 RETURN
2530 '
2540 '   PUT MCL
2550 '
2560 Q=VOICAQ+V*128
2570 POKE Q,255
2580 FOR P=Q TO Q+MLN-1
2590  POKE PTR,PEEK(P)
2600  PTR=PTR+1
2610 NEXT
2620 RETURN
2630 '
2640 '   SYNTAX ERROR
2650 '
2660 PRINT"SYNTAX ERROR in ";NUM
2670 RETURN
2680 '
2690 '   SYSTEM ERROR
2700 '
2710 PRINT"SYSTEM ERROR #";ERR
2720 END
```

リスト3-16

```
1000 ';
1010 ';   PROGRAM  MUSIC SEQUENCER
1020 ';   DATE     15-FEB-84
1030 ';
1040 '         ORG    0D000H
1050 ';
1060 'ENTRY1:  JR     START
1070 'ENTRY2:  JR     STOP
1080 'ENTRY3:  JR     CHECK
1090 ';
1100 ';   BIOS CALL
1110 ';
1120 ';   GICINI (0090H)
1130 ';     INITIALIZES PSG,AND STATIC
1140 ';     DATA FOR PLAY STATMENT
1150 ';     MODIFIES: ALL
1160 ';
1170 ';   STRTMS (0099H)
1180 ';     CHECKS AND STARTS THE BACK
1190 ';     GROUND TASK FOR PLAY
1200 ';     MODIFIES: ALL
```

```
1210 ';
1220 'GICINI  EQU   0090H
1230 'STRTMS  EQU   0099H
1240 ';
1250 ';  CONSTANT VALUES
1260 ';
1270 'MUSICF  EQU   0FB3FH
1280 'PLYCNT  EQU   0FB40H
1290 'HTIMI   EQU   0FD9FH
1300 ';
1310 ';  START MUSIC
1320 ';
1330 'START   EQU   $
1340 ';
1350 '        DI
1360 '        PUSH  AF
1370 '        PUSH  BC
1380 '        PUSH  DE
1390 '        PUSH  HL
1400 '        CALL  GICINI       ;INITIALIZE PLAY STATMENT
1410 '        LD    BC,5         ;SET HOOK TABLE
1420 '        LD    DE,HTIMI     ;INTERVAL TIMER ON
1430 '        LD    HL,HTBL1
1440 '        LDIR
1450 '        LD    BC,DTEND-DTTOP ;SET INITIAL DATA
1460 '        LD    DE,VCTBL1    ;FOR WORK AREA
1470 '        LD    HL,DTTOP
1480 '        LDIR
1490 '        LD    HL,(MCLTP1) ;SET DATA POINTER
1500 '        LD    (VCTBL1+4),HL ;FOR VOICE-A
1510 '        LD    HL,(MCLTP2) ;SET DATA POINTER-2
1520 '        LD    (VCTBL2+4),HL ;FOR VOICE-B
1530 '        LD    HL,(MCLTP3) ;SET DATA POINTER-3
1540 '        LD    (VCTBL3+4),HL ;FOR VOICE-C
1550 '        POP   HL
1560 '        POP   DE
1570 '        POP   BC
1580 '        POP   AF
1590 '        EI
1600 '        RET
1610 ';
1620 ';  STOP MUSIC
1630 ';
1640 'STOP    EQU   $
1650 ';
1660 '        DI
1670 '        PUSH  AF
1680 '        PUSH  BC
1690 '        PUSH  DE
1700 '        PUSH  HL
1710 '        LD    BC,5         ;RESET HOOK TABLE
1720 '        LD    DE,HTIMI     ;INTERVAL TIMER OFF
```

```
1730 '          LD     HL,HTBL2
1740 '          LDIR
1750 '          CALL   GICINI          ;INITIALIZE PLAY STATMENT
1760 '          POP    HL
1770 '          POP    DE
1780 '          POP    BC
1790 '          POP    AF
1800 '          EI
1810 '          RET
1820 ';
1830 ';    CHECK FLAG
1840 ';
1850 'CHECK     EQU    $
1860 ';
1870 '          PUSH   BC
1880 '          PUSH   DE
1890 '          PUSH   HL
1900 '          XOR    A
1910 '          LD     B,3
1920 '          LD     DE,4
1930 '          LD     HL,VCTBL1
1940 'CHK1:     ADD    HL,DE
1950 '          OR     (HL)            ;CHECK COUNTER
1960 '          INC    HL
1970 '          OR     (HL)
1980 '          INC    HL
1990 '          DJNZ   CHK1            ;CHECK ALL VOICE
2000 '          POP    HL
2010 '          POP    DE
2020 '          POP    BC
2030 '          AND    A               ;DATA EMPTY?
2040 '          RET    Z               ;YES,A:=0
2050 '          LD     A,0FFH          ;NO,A:=FFH
2060 '
2070 '          RET
2080 ';
2090 ';    PLAY
2100 ';
2110 'PLAY      EQU    $
2120 ';
2130 '          PUSH   AF
2140 '          PUSH   BC
2150 '          PUSH   DE
2160 '          PUSH   HL
2170 '          CALL   CHECK           ;DATA EMPTY?
2180 '          JR     Z,YALP          ;YES,RETURN
2190 '          LD     A,(MUSICF)      ;CHECK MUSICF
2200 '          OR     A               ;MUSICF=0?
2210 '          JR     NZ,YALP         ;NO,RETURN
2220 '          LD     HL,VCTBL1       ;HL:=ADR. OF VOICE-A TABLE
2230 '          CALL   PUTQUE          ;PUT DATA TO QUEUE
2240 '          LD     HL,VCTBL2       ;HL:=ADR. OF VOICE-B TABLE
```

```
2250 '          CALL   PUTQUE        ;PUT DATA TO QUEUE
2260 '          LD     HL,VCTBL3     ;HL:=ADR. OF VOICE-C TABLE
2270 '          CALL   PUTQUE        ;PUT DATA TO QUEUE
2280 '          LD     A,1           ;SET PLYCNT
2290 '          LD     (PLYCNT),A
2300 '          CALL   STRTMS        ;START MUSIC
2310 'YALP:     POP    HL
2320 '          POP    DE
2330 '          POP    BC
2340 '          POP    AF
2350 '          RET
2360 ';
2370 ';   PUT MCL INTO QUEUE
2380 ';
2390 'PUTQUE    EQU    $
2400 ';
2410 '          LD     (SAVSP),SP    ;SAVE SP
2420 '          LD     SP,HL         ;SP:=ADR. OF VOICE TABLE
2430 '          POP    BC            ;BC:=ADR. OF QUEUE TABLE
2440 '          POP    DE            ;DE:=ADR. OF QUEUE
2450 '          POP    HL            ;HL:=ADR. OF DATA
2460 '          LD     A,L           ;END OF DATA?
2470 '          OR     H
2480 '          JR     Z,EUQTUP      ;YES,RETURN
2490 '          LD     A,(HL)        ;GET SIZE OF DATA
2500 '          DEC    A
2510 '          LD     (BC),A        ;SET INTO QUEUE TABLE
2520 '          INC    BC
2530 '          XOR    A             ;CLEAR QUEUE COUNTER
2540 '          LD     (BC),A
2550 '          LD     A,(HL)
2560 '          INC    HL
2570 '          LD     C,(HL)        ;GET LINK FOR NEXT DATA
2580 '          INC    HL
2590 '          LD     B,(HL)
2600 '          INC    HL
2610 '          PUSH   BC            ;SET DATA POINTER
2620 '          LD     C,A           ;BC:=SIZE OF DATA
2630 '          LD     B,0
2640 '          LDIR
2650 '          XOR    A
2660 '          LD     (DE),A
2670 'EUQTUP:   LD     SP,(SAVSP)    ;RECOVER SP
2680 '          RET
2690 ';
2700 ';   HOOK TABLE
2710 ';
2720 'HTBL1     EQU    $
2730 ';
2740 '          RST    30H
2750 '          DEFB   0
2760 '          DEFW   PLAY
```

```
2770 '          RET
2780 ';
2790 'HTBL2     EQU     $
2800 ';
2810 '          RET
2820 '          RET
2830 '          RET
2840 '          RET
2850 '          RET
2860 ';
2870 ';     INITIAL DATA
2880 ';
2890 'DTTOP     EQU     $
2900 ';
2910 '          DEFW    0F959H          ;VOICE-A QUEUE TABLE
2920 '          DEFW    0F975H          ;VOICE-A QUEUE
2930 '          DEFW    0               ;VOICE-A DATA POINTER
2940 ';
2950 '          DEFW    0F95FH          ;VOICE-B QUEUE TABLE
2960 '          DEFW    0F9F5H          ;VOICE-B QUEUE
2970 '          DEFW    0               ;VOICE-B DATA POINTER
2980 ';
2990 '          DEFW    0F965H          ;VOICE-C QUEUE TABLE
3000 '          DEFW    0FA75H          ;VOICE-C QUEUE
3010 '          DEFW    0               ;VOICE-C DATA POINTER
3020 ';
3030 'DTEND     EQU     $
3040 ';
3050 ';     WORK AREA
3060 ';
3070 '          ORG     0F300H
3080 ';
3090 'WORK      EQU     $
3100 ';
3110 'MCLTP1:   DEFS    2               ;VOICE-A DATA ADDRESS
3120 'MCLTP2:   DEFS    2               ;VOICE-B DATA ADDRESS
3130 'MCLTP3:   DEFS    2               ;VOICE-C DATA ADDRESS
3140 ';
3150 'VCTBL1:   DEFS    6               ;VOICE-A TABLE
3160 'VCTBL2:   DEFS    6               ;VOICE-B TABLE
3170 'VCTBL3:   DEFS    6               ;VOICE-C TABLE
3180 ';
3190 'SAVSP:    DEFS    2
3200 ';
3210 '          END
```

4.ランタイム・ルーチン

ランタイム・ルーチンは，コンパイラで生成したデータを再びキューに戻し，音楽を演奏させるプログラムです。

各チャンネルのデータの先頭アドレスをそれぞれMCLTP1,MCLTP2,MCLTP3にセットし，ENTRY1をコールすると演奏が開始されます。演奏を途中で止めたいときはENTRY2をコールしてください。

音楽が演奏中であるかどうか調べたいときは，ENTRY3をコールします。演奏が終了していれば，Aレジスタの内容は0になります。

5.ソース・プログラムの形式

ソース・プログラムには，PLAY文の他にREM文，“ ' ”を使うことができます(**リスト3-17**)。ただし，当然ながらミュージック・マクロ言語の“X”，“＝”は使用することができません。

こうして作ったソース・プログラムはアスキー形式※でカセットにセーブしておきます。

> **※アスキー形式でファイルのロード・セーブをするときは，必ずカセットにリモート端子を接続してください。**

リスト3-17

```
1000 REM
1010 REM         TEST DATA
1020 REM           Bach
1030 REM   Inventions a 3 Voix 3
1040 REM
1050 PLAY"R805L16F+GL8AC04B05L16EF+L8G04B","R1","04L4DR803F+8GR8E8"
1060 PLAY"L8A05L16DEL8F+04AL16G05F+EDC+BAG","R1","03F+R8L8DBGA02A"
1070 '              :
1080 '              :
1090 '              :
1100 PLAY"L16G05F+ED04L4C+L2D","04L16F+AGF+EBAGL2F+","L802B03GA02AL2D"
```

この章ではＰＳＧの基本的な応用例をあげて解説してきましたが，工夫次第で思いのほかにおもしろい効果を得ることができます。とにかくいろいろ試してみることが一番のようで，オリジナル・サウンドを作って，あなたのゲームをより楽しいものにしましょう。

4章 キー入力

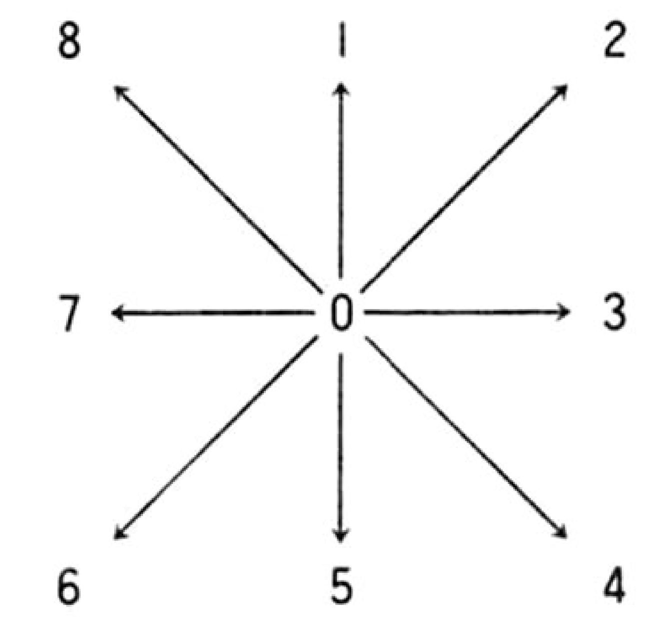
　この章ではキーボードやジョイスティックからの入力をマシン語で行うプログラムとこれまで述べなかったBIOSルーチンを解説します。

キー入力

1 キーボードとジョイスティック

リアルタイム・ゲームでは，カーソル・キーやジョイスティックの入力によって，キャラクタ・パターンを移動するのが一般的です。すでに第2章のプログラムにも登場したGTSTCKというBIOSのルーチンは，Aレジスタに装置番号を入れてコールすると**図4-1**のような数値をAレジスタに入れて戻ってきます。ここで装置番号は，

図4-1　ジョイスティックの方向



0：カーソル・キー
1：ジョイスティック1
2：ジョイスティック2

となっています。カーソル・キーでは，たとえば上と左方向のキーを押すと8が返ってきます。

これと同じようなルーチンにGTTRIGがあります。これは，スペース・キーまたはジョイスティックのトリガーボタンが押されているかどうか調べるものです。Aレジスタに装置番号（0ならスペース・キー，1と2はそれぞれジョイスティック1と2）を入れてコールすると，もし押されていれば255 (FFH) を，押されていなければ0が返ってきます。

この他にリアルタイムに複数のキー入力を調べるためにBIOSにはSNSMATというルーチンがあります。このルーチンを使うと一度に8個のキー入力を調べることができます。MSXでは**図4-2**のようにキーをグループ分けしています。

たとえば，[Z]と[X]が押されているかどうかはグループ5だけを調べればよいので，1度SNSMATルーチンをコールするだけでよいのですが，[B]と[J]の状態を知るためにはそれぞれグループ2とグループ3を調べるため，SNSMATを2回コールしなければなりません。

SNSMATのアドレスは0141Hで，調べたいグループの番号をAレジスタにいれて，コールすると押されているキーのビットは0になり，押されていないキーのビットは1となります。この結果をAレジスタ

にいれて戻ってきます。たとえば，RETURNキーが押されているかどうかを調べるにはAレジスタにRETURNキーのグループ番号の7をいれてSNSMATをコールします。そして，Aレジスタの最上位ビットが0なら押されている，1なら押されていないと判断できます。なお，SNSMATルーチンをコールした後，AFレジスタとCレジスタの内容が変化します。

SNSMATの例として[I]と[M]で右側の，[W]と[Z]で左側のラケットを上下に動かすプログラムを作ってみました（**リスト4-1**）。

プログラムではまず，スクリーンをグラフィックIIモードにして，スプライトのパターンを定義し，右側のラケットの処理，左側のラケットの処理をします。次にCTRLキーとSTOPキーが押されているかどうかを調べます。これはSNSMATルーチンでも調べることができますが，BIOSのBREAKXルーチンを使う方が簡単です。このルーチンをコールしたときに，CTRLキーとSTOPキーを同時に押しているとキャリーフラグがたって戻ってきます。押されていないときは，ま

図4-2　SNSMATのキーマトリックス

グループNo. ＼ ビット	7	6	5	4	3	2	1	0
0	7	6	5	4	3	2	1	0
1	;	[	@	¥	^	−	9	8
2	B	A	_	／	.	,	]	:
3	J	I	H	G	F	E	D	C
4	R	Q	P	O	N	M	L	K
5	Z	Y	X	W	V	U	T	S
6	F3	F2	F1	かな	CAPS	GRAPH	CTRL	SHIFT
7	RETURN	SELECT	BS	STOP	TAB	ESC	F5	F4
8	→	↓	↑	←	DEL	INS	HOME	SPACE

たラケットの処理へ戻ります。ラケットの処理ではSNSMATルーチンを使って，どのキーが押されたかを判断し，それによって，ラケットのY座標を変化させています。

このプログラムにボールの移動を処理するプログラムを加えれば，2人で遊べるテニス・ゲームになります。トライしてみてください。

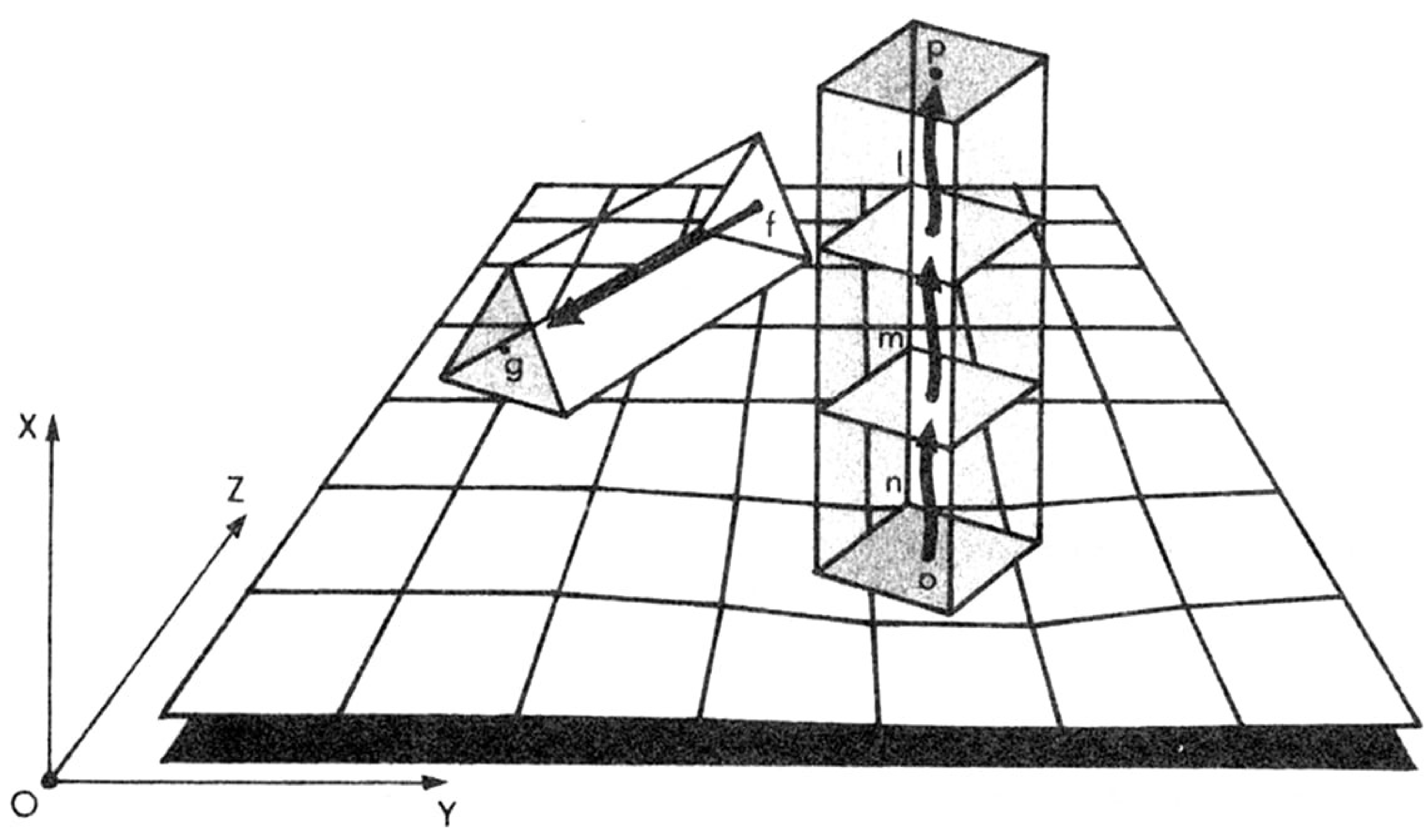
リスト4-1

```
1000 ';
1010 ';   PROGRAM USE SNSMAT
1020 ';   DATE    27-JUL-84
1030 ';
1040 '         ORG   0D000H
1050 ';
1060 ';   BIOS CALL
1070 ';
1080 ';   WRTVDP (0047H)
1090 ';     WRITES TO VDP
1100 ';     ENTRY:   REGISTER# => C
1110 ';              DATA => B
1120 ';     RETURNS: NONE
1130 ';     MODIFIES:AF,BC
1140 ';   WRTVRM (004DH)
1150 ';     WRITES TO VRAM
1160 ';     ENTRY:   VRAM ADDRESS => HL
1170 ';              DATA => A
1180 ';     RETURNS: NONE
1190 ';     MODIFIES:AF
1200 ';   LDIRVM (005CH)
1210 ';     TO VRAM FROM MEMORY
1220 ';     ENTRY:   SOURCE ADDR => HL
1230 ';              DESTINATION => DE
1240 ';              LENGTH => BC
1250 ';     RETURNS: NONE
1260 ';     MODIFIES:ALL
1270 ';   INIGRP (0072H)
1280 ';     INITIALIZE FOR GRAPHIC
1290 ';     ENTRY:   GRPNAM (0F3C7H)
1300 ';              GRPCOL (0F3C9H)
1310 ';              GRPCGP (0F3CBH)
1320 ';              GRPATR (0F3CDH)
1330 ';              GRPPAT (0F3CFH)
1340 ';     RETURNS: NONE
1350 ';     MODIFIES:ALL
1360 ';   BREAKX (00B7H)
1370 ';     CHECKS THE STATUS OF
1380 ';     CTRL-STOP KEY
1390 ';     ENTRY:   NONE
```

```
1400 ';     RETURNS: IF BEING PRESSED
1410 ';              THEN CY <= 1
1420 ';     MODIFIES:AF
1430 ';   SNSMAT (0141H)
1440 ';     SENSE KEY MATRIX
1450 ';     ENTRY:   LINE# => A
1460 ';     RETURNS: BIT PATTERN => A
1470 ';     MODIFIES:AF,C
1480 ';
1490 'WRTVDP  EQU   0047H
1500 'WRTVRM  EQU   004DH
1510 'LDIRVM  EQU   005CH
1520 'INIGRP  EQU   0072H
1530 'BREAKX  EQU   00B7H
1540 'SNSMAT  EQU   0141H
1550 'SPRITE  EQU   1B00H
1560 'PAT     EQU   3800H
1570 ';
1580 ';   START
1590 ';
1600 'START   EQU   $
1610 ';
1620 '        CALL  INIVDP   ;SET SCREEN MODE
1630 'LOOP:   CALL  RIGHT    ;RIGHT RACKET CONTROL
1640 '        CALL  LEFT     ;LEFT RACKET CONTROL
1650 '        LD    BC,0400H ;WAIT
1660 'DELAY:  DEC   BC
1670 '        LD    A,B
1680 '        OR    C
1690 '        JR    NZ,DELAY
1700 '        CALL  BREAKX   ;STOP KEY CHECK
1710 '        JR    NC,LOOP
1720 '        RET
1730 ';
1740 ';   RIGHT
1750 ';
1760 'RIGHT   EQU   $
1770 ';
1780 '        LD    A,(RY)
1790 '        LD    B,A      ;B:=RIGHT RACKEY POINT Y
1800 '        LD    A,3
1810 '        CALL  SNSMAT
1820 '        BIT   6,A      ;PRESS 'I'?
1830 '        JR    NZ,R1
1840 '        DEC   B        ;THEN DEC Y
1850 '        JR    R2
1860 'R1:     LD    A,4
1870 '        CALL  SNSMAT
```

```
1880 '          BIT    2,A        ;PRESS 'M'?
1890 '          JR     NZ,R2
1900 '          INC    B          ;THEN INC Y
1910 'R2:       LD     A,B
1920 '          LD     (RY),A     ;SET NEW Y
1930 '          LD     HL,SPRITE
1940 '          CALL   WRTVRM     ;WRITE NEW Y TO VRAM
1950 '          RET
1960 ';
1970 ';   LEFT
1980 ';
1990 'LEFT      EQU    $
2000 ';
2010 '          LD     A,(LY)
2020 '          LD     B,A        ;B:=LEFT RACKET Y
2030 '          LD     A,5
2040 '          CALL   SNSMAT
2050 '          BIT    4,A        ;PRESS 'W'?
2060 '          JR     NZ,L1
2070 '          DEC    B          ;THEN DEC Y
2080 '          JR     L2
2090 'L1:       BIT    7,A        ;PRESS 'Z'?
2100 '          JR     NZ,L2
2110 '          INC    B          ;THEN INC Y
2120 'L2:       LD     A,B
2130 '          LD     (LY),A     ;SET NEW Y
2140 '          LD     HL,SPRITE+4
2150 '          CALL   WRTVRM     ;WRITE NEW Y TO VRAM
2160 '          RET
2170 ';
2180 ';   INIVDP
2190 ';
2200 'INIVDP    EQU    $
2210 ';
2220 '          CALL   INIGRP     ;SET G2MODE
2230 '          LD     BC,0E201H;SPRITE SIZE 16×16
2240 '          CALL   WRTVDP
2250 '          LD     BC,8
2260 '          LD     HL,RY
2270 '          LD     DE,SPRITE;INIT SPRITE ATR
2280 '          CALL   LDIRVM
2290 '          LD     BC,32
2300 '          LD     HL,PATTBL;INIT SPRITE PATTERN
2310 '          LD     DE,PAT
2320 '          CALL   LDIRVM
2330 '          RET
2340 ';
2350 ';   DATA
```

```
2360 ';
2370 'RY:      DEFB  70H       ;Y
2380 '         DEFB  0E0H      ;X
2390 '         DEFB  0         ;SPRITE#
2400 '         DEFB  8         ;COLOR
2410 ';
2420 'LY:      DEFB  70H       ;Y
2430 '         DEFB  10H       ;X
2440 '         DEFB  0         ;SPRITE#
2450 '         DEFB  8         ;COLOR
2460 ';
2470 'PATTBL:  DEFB  0FFH,0FFH,0FFH,0FFH
2480 '         DEFB  0FFH,0FFH,0FFH,0FFH
2490 '         DEFB  0FFH,0FFH,0FFH,0FFH
2500 '         DEFB  0FFH,0FFH,0FFH,0FFH
2510 '         DEFB  0FFH,0FFH,0FFH,0FFH
2520 '         DEFB  0FFH,0FFH,0FFH,0FFH
2530 '         DEFB  0FFH,0FFH,0FFH,0FFH
2540 '         DEFB  0FFH,0FFH,0FFH,0FFH
```



キー入力

2 BIOSルーチン

ここでは本章で使ったBIOSルーチンやゲーム制作に使えそうなBIOSルーチンを説明します。

1.入力関係のBIOSルーチン

①GTSTCK(00D5H)

Aレジスタにジョイスティックの番号を入れて，このルーチンをコールすると，ジョイスティックの向いている方向をAレジスタに入れて戻ります。このルーチンをコールした後，すべてのレジスタの内容は保証されません。

②GTTRIG(00D8H)

Aレジスタにジョイスティックの番号を入れて，このルーチンをコールすると，ジョイスティックのトリガーボタンが押されているときは255を，押されていないときは0をAレジスタに入れて戻ります。このルーチンをコールした後，AFレジスタ以外のレジスタは変化しません。

③CHGET(009FH)

キーボードから入力があるまで待ち，入力した文字をAレジスタにいれて戻ります。このルーチンをコールした場合はAFレジスタの内容が変更されます。

④PINLIN(00AEH)

このルーチンはキーボードから文字列を入力するためのもので，RETURNキーまたはSTOPを入力するまでの文字列がバッファに入ります。このときHLレジスタにバッファの先頭番地－1が入ります。また，STOPを入力したときのみ，キャリーフラグがセットされます。このルーチンをコールした後は，すべてのレジスタの内容が破壊されます。

⑤QINLIN(00B4H)

このルーチンをコールすると“?”を表示して，キーボードから文字列を入力します。使い方は，PINLINルーチンと同じです。

⑥BREAKX(00B7H)

このルーチンをコールすると，CTRLとSTOPキーを押したかどうかを調べ，押していればキャリーフラグがセットされます。このルーチンをコールした後，AFレジスタ以外のレジスタは変化しません。

2. 出力関係，その他のBIOSルーチン

①CHPUT (00A2H)

Aレジスタに ASCII コードを入れて，このルーチンをコールすると画面に1文字表示します。このルーチンをコールした後，レジスタの内容は変化しません。

②CLS (00C3H)

Zフラグを立てて，このルーチンをコールすると画面をクリアします。このルーチンをコールした後は，AF，BC，DEレジスタの内容は保障されません。

③POSIT (00C6H)

Hレジスタに水平位置，Lレジスタに垂直位置を入れて，このルーチンをコールするとカーソルが移動します。コールした後，AFレジスタが破壊されます。

④BEEP (00C0H)

このルーチンをコールするとブザーを鳴らします。コールした後，すべてのレジスタの内容は保証されません。

⑤LPTOUT (00A5H)

Aレジスタに ASCII コードを入れて，このルーチンをコールするとプリンタに1文字出力します。プリンタが使えない状態であれば，キャリーフラグを立てて戻ってきます。

コールした後，フラグ以外のレジスタは変化しません。

⑥LPTSTT (00A8H)

このルーチンはプリンタの状態をチェックするためのもので，コールした後，Aレジスタが255でZフラグが立っていなければプリンタを使える状態にあることを意味します。このルーチンをコールした後，AFレジスタ以外のレジスタは変化しません。

⑦DSPFNK (00CFH)

このルーチンをコールすると，ファンクション・キーの表示をします。すべてのレジスタの内容は保証されません。

⑧ERAFNK (00CCH)

このルーチンをコールすると，ファンクション・キーの表示を消します。すべてのレジスタの内容は保証されません。

⑨CHGCAP (0132H)

Aレジスタに0を入れて，このルーチンをコールするとCAPランプがつきます。Aレジスタが0でないとランプを消します。コールした後，AFレジスタ以外のレジスタは変化しません。

⑩CHGSND（0135H）

Ａレジスタに０を入れて，このルーチンをコールするとサウンド・ポートの１ビットをオフにします。Ａレジスタが０でないとビットをオンにします。コールした後，ＡＦレジスタ以外のレジスタは変化しません。

⑪RDVDP（013EH）

このルーチンをコールするとＶＤＰのステータス・レジスタの値をＡレジスタに入れて戻ります。コールした後，Ａレジスタ以外のレジスタは変化しません。

5章 ランダム・テクニック
音声合成にチャレンジ

本章では，ランダム・テクニックとして，“乱数の生成”と“音声合成”の2つを取り上げます。両方ともゲーム以外に応用できるものです。

ランダム・テクニック

1 乱数の発生

乱数はゲームでは割と良く使われるものです。もともとコンピュータ・ゲームは，スタートレック（BASICが普及しはじめた頃非常に流行したゲーム）のように乱数を使ったシミュレーションに起源があると考えられます。もちろん今でもほとんどのゲームが乱数の要素を何らかの形で取り入れています。したがって，乱数はゲームの必需品と言ってもいいでしょう。

BASICではRNDという関数があり，これを使って乱数を発生させることができますが（コンピュータで発生させられるのは正式には疑似一様乱数です。ここでは便宜上乱数として扱います），機械語では乱数を自分で作らなくてはなりません。

そこで乱数を発生させる例として**リスト5-1**のプログラムを紹介します。このルーチンを呼ぶとAレジスタに乱数が入ります。このルーチンではZ80のリフレッシュ・レジスタを利用して乱数を発生させています。このリフレッシュ・レジスタは，ダイナミックRAMのリフレッシュ・アドレスを発生させるために使われているもので，ソフト的にはほとんど利用することがありません。Z80では乱数発生のためだけに利用されているようです。もちろんリフレッシュ・レジスタの内容をそのまま乱数として利用するわけにはいかないので，その値に多少手を加えます。これでとりあえず使用に耐えうる乱数（疑似一様乱数）が出来上ります。

乱数の発生法には，このほかにもいくつかのやり方があるので，各自工夫してみてください。

リスト5-1

```
1000 ';
1010 ';   PROGRAM    RANDOM
1020 ';   DATE       6-FEB-84
1030 ';
1040 '         ORG   0D000H
1050 ';
1060 ';   GENERATE RANDOM NUMBER
1070 ';
1080 ';    ENTRY:    NONE
1090 ';    RETURNS:  NUMBER => A
```

```
1100 ';    MODIFIES: AF
1110 ';
1120 'RANDOM  EQU   $
1130 ';
1140 '        PUSH  BC
1150 '        LD    BC,(RND)       ;GET OLD NUMBER
1160 '        LD    A,R            ;GET REFRESH REGISTER
1170 '        ADD   A,C            ;ADD OLD NUM1
1180 '        ADD   A,B            ;ADD OLD NUM2
1190 '        LD    C,B            ;SAVE NUMBER
1200 '        LD    B,A
1210 '        LD    (RND),BC
1220 '        POP   BC
1230 '        RET
1240 ';
1250 ';   WORK AREA
1260 ';
1270 'RND:    DEFW  0AA55H
1280 ';
1290 '        END
```

2 ランダム・テクニック

1ビットサウンド・ポートによる音声合成

ＭＳＸでは，ＰＳＧを使わずにソフトウェアでサウンドを直接合成することができます。といってもこれは単に１ビットのポート出力をソフトウェアでコントロールして，方形波を作り出すという単純なものです。したがって，一定の周波数の音やノイズを出すといった場合にはＰＳＧの能力にはかないませんが，逆により自由度の高い利用法が考えられます。そこで，ここではこのサウンド・ポートを利用して，音声の合成をしてみることにしましょう。

音声合成の原理については説明するとかなりの量になるのでここでは省略します。一方プログラムの方はというと，これは実に簡単で，**リスト5-2** のようにごく短いものです。このプログラムには２つの機能があり，カセット入力ポートから音声を読みこんでメモリにストアする(録音)，メモリ中の音声データを１ビット・サウンド・ポートに出力する（再生）といったことができます。記録できる時間は４秒程度です。

使い方は次のとおりです。

●録音

1) 合成させたい音声をカセットテープに録音する。

2) テープを再生させながら，プログラム中のENTRY 1 をコールする。再生レベルは高めの方がよい。

●**再生**

1) プログラム中のENTRY 2 をコールする。

音質に不満は残りますが，このような簡単なプログラムでも何とか音声を合成することができます。欠点はメモリを大量に消費すること(数秒間の合成で 4 ～16 K バイトのメモリが必要）でしょう。

図5-1 フローチャート(録音時)

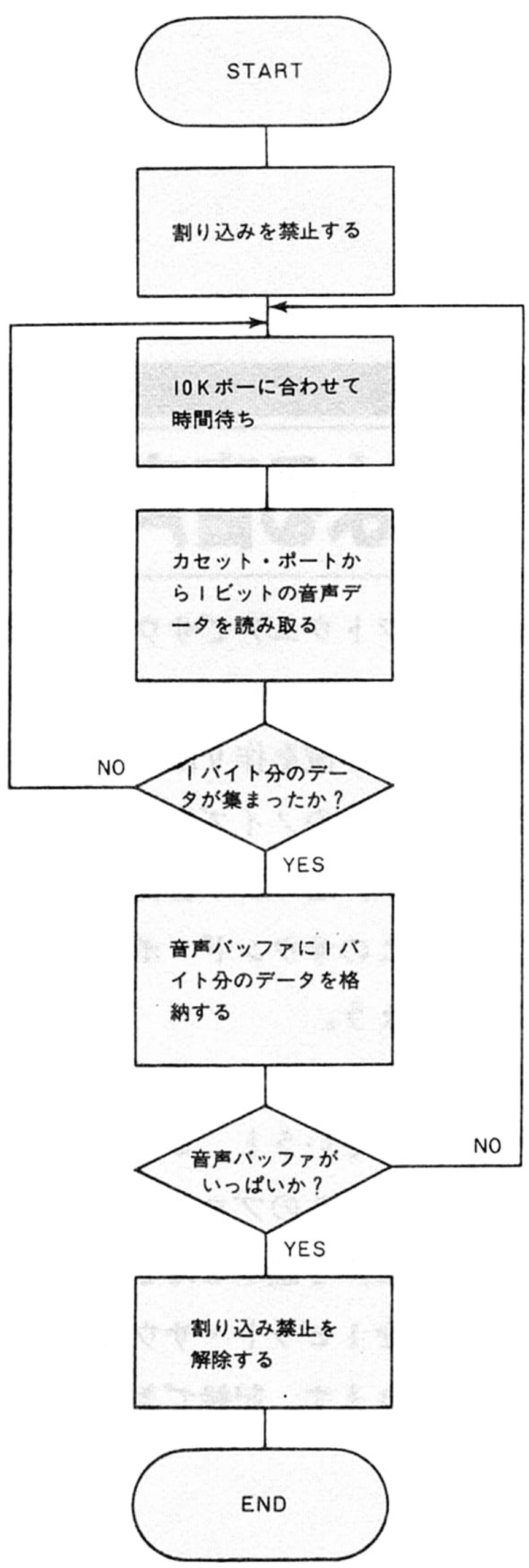
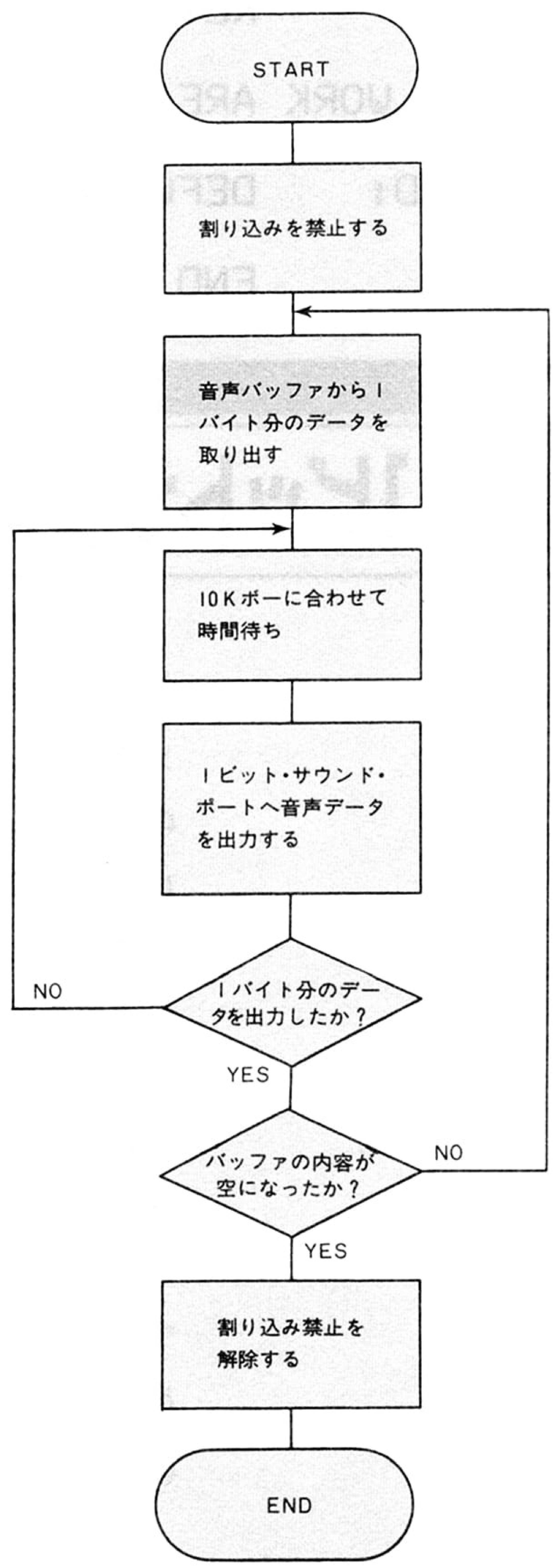


図5-2 フローチャート(再生時)



リスト5-2

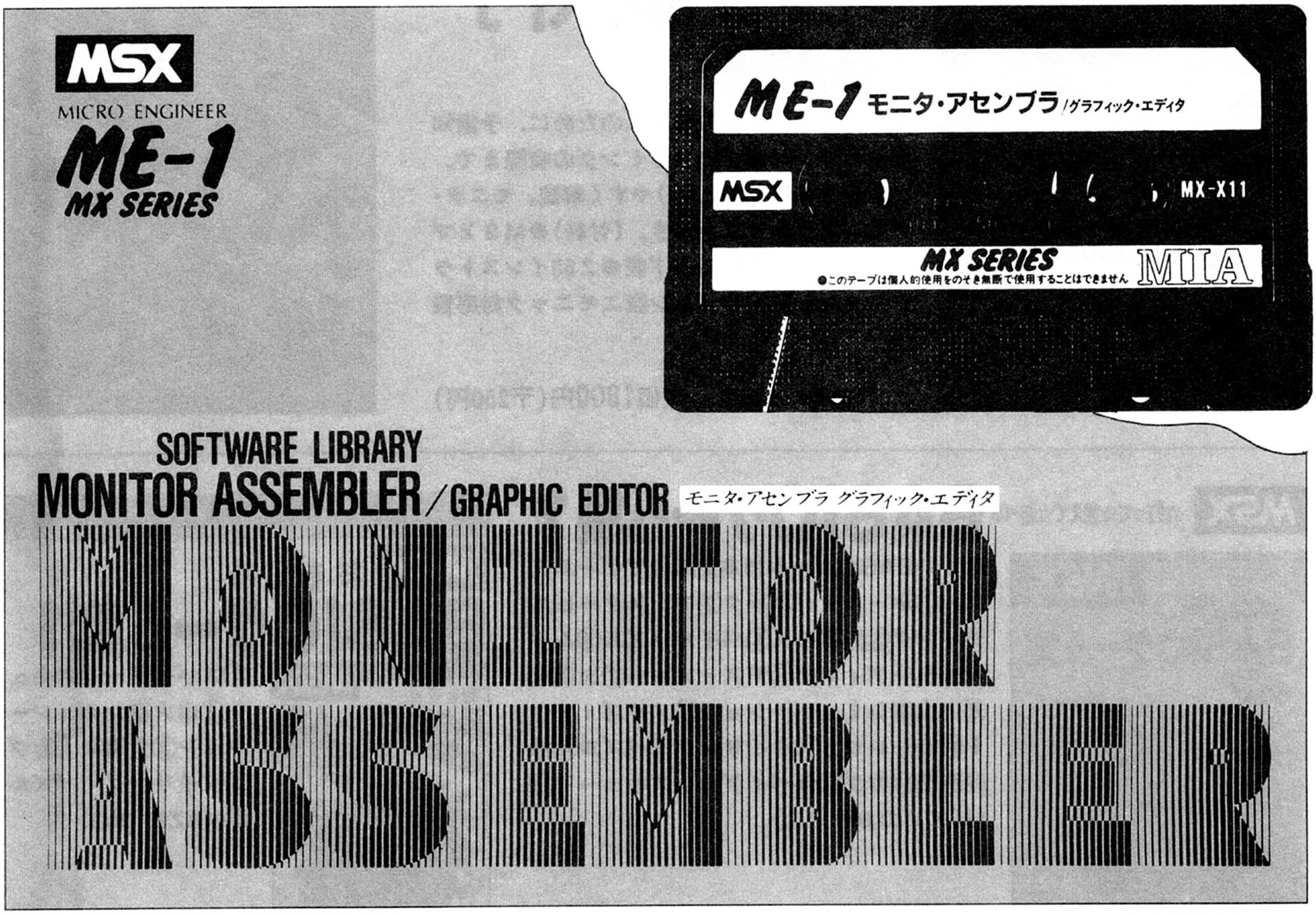
```
1000 ';
1010 ';   PROGRAM  MSX-RECORDER
1020 ';   DATE     12-FEB-84
1030 ';
1040 '          ORG    0D000H
1050 'ENTRY1: JR     RECORD    ;TO RECORDER
1060 'ENTRY2: JR     PLAY      ;TO PLAYER
1070 ';
1080 ';   BIOS CALL
1090 ';
1100 ';   RDPSG  (0096H)
1110 ';     READS A DATA FROM PSG REG.
1120 ';     ENTRY:    Acc < REG. NO.
1130 ';     RETURNS:  DATA>Acc
1140 ';     MODIFIES: NONE
1150 ';
1160 ';   CHGSND (0135H)
1170 ';     CHANGES THE STATUS OF
1180 ';     1 BIT SOUND PORT
1190 ';     ENTRY:    Acc<0 TO OFF
1200 ';               Acc<1..255 TO ON
1210 ';     RETURNS:  NONE
1220 ';     MODIFIES: NONE
1230 ';
1240 'RDPSG   EQU    0096H
1250 'CHGSND  EQU    0135H
1260 ';
1270 ';   CONSTANT VALUES
1280 ';
1290 'BUFF    EQU    0D800H    ;ADDRESS OF BUFFER
1300 'BFSIZE  EQU    1000H     ;BUFFER SIZE
1310 ';
1320 ';   RECORDER
1330 ';
1340 'RECORD  EQU    $
1350 ';
1360 '        DI               ;INTERRUPT MUST BE DISABLED
1370 '        LD     HL,BUFF   ;(HL):=ADDRESS OF BUFFER
1380 '        LD     DE,BFSIZE;(DE):=BUFFER SIZE
1390 'REC1:   LD     C,8
1400 'REC2:   CALL   DELY      ;DELAY
1410 '        LD     A,14
1420 '        CALL   RDPSG     ;GET DATA BIT FROM PORT
1430 '        RL     A
1440 '        RL     B
1450 '        DEC    C         ;DONE WITH BYTE
1460 '        JR     NZ,REC2   ;NO,CONTINUE
```

```
1470 '           LD    (HL),B    ;PUT DATA BYTE TO BUFFER
1480 '           INC   HL
1490 '           DEC   DE        ;BUFFER FULL?
1500 '           LD    A,E
1510 '           OR    D
1520 '           JR    NZ,REC1   ;NO,CONTINUE
1530 '           EI
1540 '           RET
1550 ';
1560 ';   PLAYER
1570 ';
1580 'PLAY       EQU   $
1590 ';
1600 '           DI              ;INTRRUPT MUST BE DISABLED
1610 '           LD    HL,BUFF   ;(HL):=ADDRESS OF BUFFER
1620 '           LD    DE,BFSIZE;(DE):=BUFFER SIZE
1630 'PLY1:      LD    B,(HL)    ;GET DATA BYTE FROM BUFFER
1640 '           INC   HL
1650 '           LD    C,8
1660 'PLY2:      CALL  DELY      ;DELAY
1670 '           XOR   A
1680 '           RL    B
1690 '           RL    A
1700 '           CALL  CHGSND    ;PUT DATA BIT TO SOUND PORT
1710 '           DEC   C         ;DONE WITH BYTE?
1720 '           JR    NZ,PLY2   ;NO,CONTINUE
1730 '           DEC   DE        ;BUFFER EXHAUSTED?
1740 '           LD    A,E
1750 '           OR    D
1760 '           JR    NZ,PLY1   ;NO,CONTINUE
1770 '           EI
1780 '           RET
1790 ';
1800 'DELY:      LD    A,12
1810 'DEL1:      DEC   A
1820 '           JP    NZ,DEL1
1830 '           RET
1840 ';
1850 '           END
```

MSX INFORMATION

頼れるツール、登場!!

ME マイクロ エンジニア MICRO ENGINEER



ME-1

MSX モニタ・アセンブラ

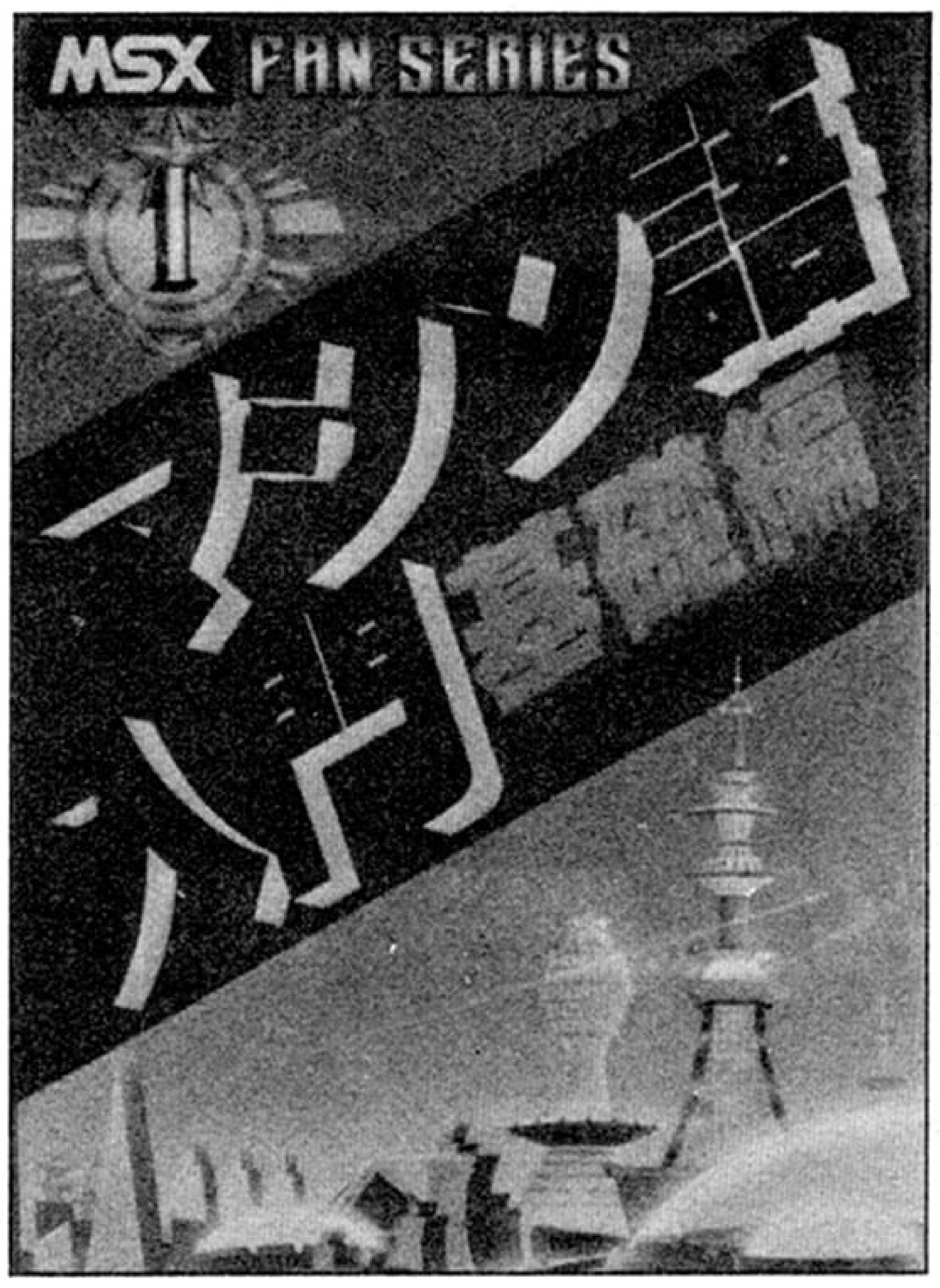
マシン語プログラムの制作に必要なアセンブラ。テキスト・サーチ機能，擬似命令，チェック・サム計算，オン・メモリでのアセンブル…何でもできる高機能と使いやすさ。雑誌等に掲載のマシン語ゲームを打ち込むのに不可欠なモニタの付いた「ＭＳＸモニタ・アセンブラ」です。（グラフィック機能をフルに引き出すグラフィック・エディタ付）

（テープ，マニュアル付）定価**4800円**（〒240円）

●お求めは最寄りのマイコン・ショップへ,, または郵送料を添えて下記へお申し込み下さい。

〒150 東京都渋谷区渋谷2-9-1青山田中ビル TEL(03)486-4500(代表) ㈱エム・アイ・エー **MIA**

MSX INFORMATION



MSX FANシリーズ1
マシン語入門 (基礎編)

ＭＳＸでマシン語を学ぶ人のために，予備知識からマシン語プログラミングの実際まで、図表等を多用してわかりやすく解読。モニタ・アセンブラ全リスト付き。(付録)●ＭＳＸマシンのキャラクタ・コード表●Ｚ80インストラクション一覧表●マシン語ニモニック対応表

B5判　160頁　定価1800円(〒250円)



MSX ☆打って☆覚えて☆遊べる BASICゲーム集1



楽しいBASICゲーム15本を集めてゲーム集に。①ホール・パニック②モンスタービルディング③5-ダイス④バイオリズム⑤ムーン・ランディング⑥デス・スキー⑦大海戦⑧山火事シミュレーション⑨メイズ・アウト⑩ルーレット⑪タイリング・パズル⑫神経衰弱⑬カブ⑭スパイダーレスキュー　⑮(ピアノのおけいこ)

A5判　定価1500円(〒250円)

テープ版　6本入り

MSX おもしろ BASICゲーム集1



①スパイダーレスキュー②山火事シミュレーション③大海戦④カブ⑤タイリング・パズル⑤MAZE OUT

定価3000円(〒350円)

MSX ☆打って☆覚えて☆遊べる BASICゲーム集3



打ってワクワク，遊んでドキドキ，期待のBASICゲーム集第３弾。①ジョギングの邪魔はしないで②GO！GO！SLOT③蛇の道はHeavy④恐怖の立体迷路⑤わんぱくネコちゃん大奮闘⑥ああ青春アメフト野郎⑦３目並べ本因坊戦⑧電子式ハノイの塔⑨ジェット・ヘリ空中戦⑩３次元エレガント・スカッシュ⑪ブラック・ジャック⑫WANTED!!馬泥棒⑬ボクはクレーン操縦士⑭ソリティア遊び⑮恋とバイオリズム相性診断

A5判　定価1500円(〒250円)

テープ版　6ゲーム入り

MSX おもしろ BASICゲーム集3



①ジョギングの邪魔はしないで②三次元エレガント・スカッシュ③ジェット・ヘリ空中戦④ああ青春アメフト野郎⑤恐怖の立体迷路⑥恋とバイオリズム相性診断

定価3000円(〒350円)

MSX FANシリーズ3
マシン語入門 (実践編)

マシン語の予備知識を得，実際にプログラミングにかかろうという人のハンドブックとして最適。初心者が陥りやすいプログラミングの落とし穴をすべてフォローした基本テクニック集。●何はともあれプログラムしてみよう●マシン語の定石●基本テクニックのまとめ●メインディッシュ(実践テクニック)●応用

B5判　180頁　定価1800円(〒250円)

MSX 快速マシン語ゲーム集



人気のMSX用マシン語ゲームを集めて全リスト公開。BASICとは違い，ハードウェアの機能を十二分に引き出した高速ゲーム。さらにマシン語リストの打ち込み方法やマシン語モニタを平易に解説。①おてんばベッキーの大冒険②ファイナル麻雀③ニョロルス④アドベン・チュー太⑤ジャンピング・ラビット⑥ジグソーセット⑦ロンサム・タンク進撃

A5判　定価1500円(〒250円)

MSX ☆打って☆覚えて☆遊べる
BASICゲーム集2



大好評のBASICゲーム集の第2弾。よりおもしろく，楽しく。①スーパー光線砲迎撃部隊②宇宙人が降ってくる日③すペーす・くらんぼー④ちんちろ遊び⑤ストン・ボール⑥ザ・コンバート・ポーカー⑦超能力モンキーVSゴロツキ虫⑧インベリアンくずし⑨スネーク・ハンター⑩HOLE DOWN⑪ニコニコ風船は圧死の運命⑫宇宙要塞Z1007爆破作戦

A5判　定価1500円(〒250円)

テープ版　6ゲーム入り

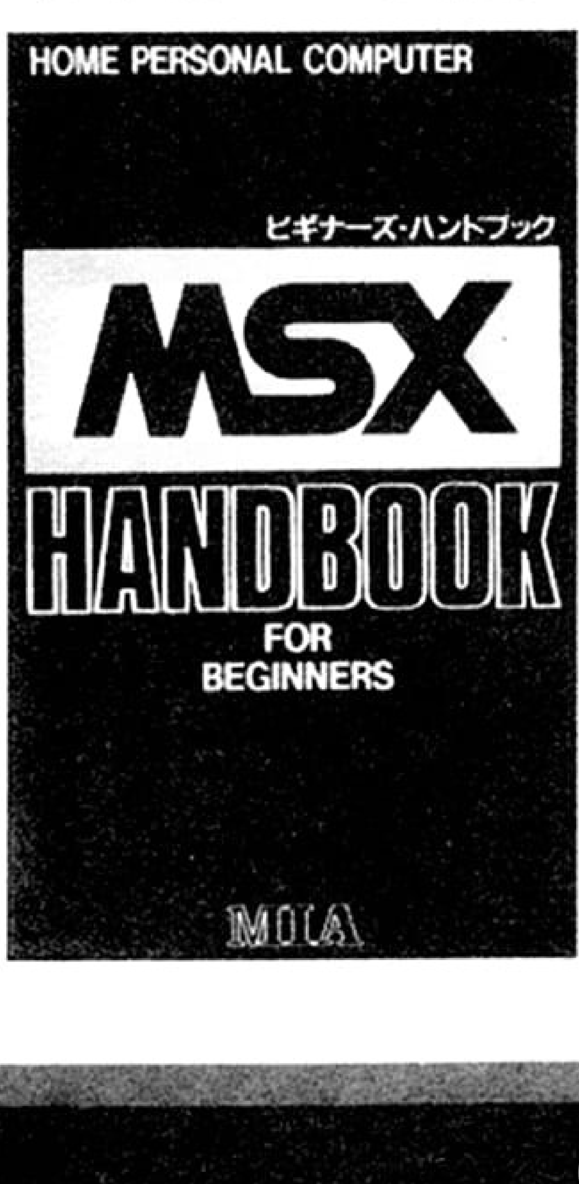
MSX おもしろ
BASICゲーム集2



①スーパー光線砲迎撃部隊②ちんちろ遊び③超能力モンキーVSゴロツキ虫④インベリアンくずし⑤HOLE DOWN⑥ニョロルス

定価3000円(〒350円)

MSX ビギナーズ・ハンドブック
1台に1冊！ ホームパソコン雑学まるかじり事典

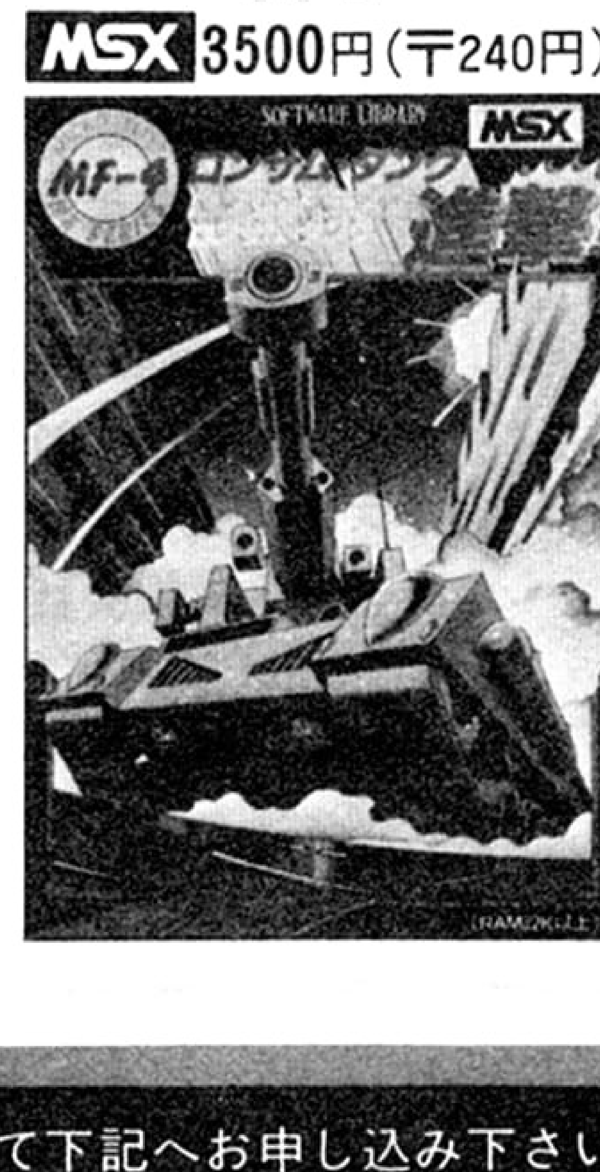


取扱説明書も，マニュアルも，入門書を読んでもよくわからない「パソコン用語のあれこれ」をイラストをまじえてやさしく解説。MSXの初歩的な操作ポイントや各種のトラブル対策から，学習のコツ，ホビーの楽しみかた，より有効的な活用法，パソコンに関する知恵・知識・用語までを多彩にツメ込んでみた。資料編（エラーがでたときのチェック・ポイント，機種別索引，メモリ・マップ，キャラクタ・コード表…他）

新書版　定価980円(〒200円)

MIAのカセットテープ

ロンサム・タンク新撃
MSX 3500円(〒240円)



ジャンピング・ラビット
MSX 3500円(〒240円)



MSX FANシリーズ2 **マシン語入門〔応用編〕**

1984年 9 月 1 日　初版発行
1985年 4 月 10日　第 2 刷発行
著　　者　白井康之
編　　集　MIA第2編集部
発 行 人　塚本慶一郎
発 行 所　株式会社 エム・アイ・エー
〒150 東京都渋谷区渋谷2-9-1青山田中ビル
電話(03)486-4500（代）
イラスト　大村敦郎
印刷・製本　有限会社 チトセ印刷

ISBN4-87170-023-2 C3055 ¥1800E

料金受取人払

郵便はがき

渋谷局承認

2444

差出有効期間
昭和60年12月
31日まで

150-□□

東京都渋谷区渋谷二―九―一
青山田中ビル

㈱エム・アイ・エー
マシン語入門(応用編)
編集部 係

フリガナ		性別	男・女
氏　名		年令	才
ご住所	〒 Tel.(　　)		
勤務先・所属部課 学校名・学科名			
お持ちのマイコン		使用するプログラム言語	
マイコン・クラブ		経験	年

アンケート

本誌をお買いあげいただきありがとうございました。みなさんのご意見を編集の参考にさせていただきたいと思います。アンケートにご協力ください。

1 本誌を何でお知りになりましたか。

●書店〔　　〕　●広告〔　　〕　●友人，知人〔　　〕
●パソコンショップ〔　　〕　●その他〔　　　　〕
●店名〔　　　　　　　　〕

2 本誌をどこでお買いになりましたか。

●書店〔　　〕　●パソコンショップ〔　　〕
●その他〔　　　　　　　　〕
●店名〔　　　　　　　　〕

3 本誌の内容はいかがでしたか。感想をお書きください。

〔　　　　　　　　　　　　　　　　〕

4 本誌内容の編集部へのご意見・ご希望をお聞かせください。

〔　　　　　　　　　　　　　　　　〕

5 今後とりあげてほしい企画がありましたらお書きください。

〔　　　　　　　　　　　　　　　　〕

MIA COMPUTER BOOKS

MSX fan series

MACHINE LANGUAGE

MIA

MICRO INFORMATION ASSOCIATES

ISBN4-87170-023-2 C3055 ¥1800E 定価1800円